VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Řízení IP telefonních toků s využitím technologie softwarově definovaných sítí

Control of IP Telephony Flows Using Software-defined Network Technology

Bc. Dalibor Zeman

Diplomová práce

Vedoucí práce: Ing. Filip Řezáč, Ph.D.

Ostrava, 2021

Abstrakt

Cílem této diplomové práce je navrhnout a implementovat mechanismus adaptivně konfigurující QoS pravidla na základě informací poskytovaných protokolem SIP. Koncept SDN sítí příhodně přináší nové možnosti v oblasti real-time konfiguraci na centralizované bázi. Jako SIP server bylo zvoleno Kamailio, které SDN kontroleru Ryu předává informace o probíhajících hovorech, implementující požadovanou konfiguraci. Po implementaci následuje otestování navrženého systému, které potvrzuje jeho funkčnost a pozitivní vliv na kvalitativní parametry hovorů.

Klíčová slova

QoS; SDN; SIP; VoIP; Kamailio

Abstract

The main objective of this thesis is to design and implement a mechanism adaptively configuring QoS policies based on information provided by the SIP protocol. Conveniently, the SDN concept opens up new possibilities in terms of real-time configuration on a centralized basis. Kamailio has been chosen as a SIP server, handing desired information over to the Ryu SDN controller. After the implementation, a comparison of QoS parameters with and without the designed system follows. The results validate the functionality of the designed system and its positive impact on QoS parameters of test calls.

Keywords

QoS; SDN; SIP; VoIP; Kamailio

Poděkování

Chtěl bych poděkovat Ing. Filipu Řezáčovi Ph.D. za dlouholetou spolupráci a benevolentní přístup k mé někdy trochu svévolné pracovní morálce. Speciální poděkování patří mé partnerce Lindě, která mi po čas psaní této práce byla neocenitelnou psychickou oporou.

Obsah

Seznam použitých zkratek a symbolů

ACL	–	Access-Control List
API	–	Application Programming Interface
B2BUA	–	Back-to-Back User Agent
CBWFQ	–	Class-Based Weighted Fair Queuing
CLI	–	Command-Line interface
CQ	–	Custom Queuing
FIFO	–	First In First Out
HTTP	–	Hypertext Transfer Protocol
IDS	–	Intrusion Detection System
IETF	–	Internet Engineering Task Force
IP	–	Internet Protocol
LLQ	–	Low Latency Queuing
MPLS	–	Multiprotocol Label Switching
NAT	–	Network Address Translation
OVS	–	Open vSwitch
PHB	–	Per-Hop Behavior
PQ	–	Priority Queuing
QoS	–	Quality of Service
RFC	–	Request For Comments
RTP	–	Real-time Transport Protocol
SDN	–	Software Defined Networks
SDP	–	Session Description Protocol
SIP	–	Session Initation Protocol
SNMP	–	Simple Network Management Protocol
SRTP	–	Secure Real-time Transport Protocol
TCP	–	Transmission Control Protocol
TLS	–	Transport Layer Security
UAC	–	User Agent Client

UAS	–	User Agent Server
UA	–	User Agent
URI	–	Uniform Resource Identifier
VoIP	–	Voice over Internet Protocol
WFQ	–	Weighted Fair Queuing

Seznam obrázků

Kapitola 1

Úvod

U IP telefonních komunikací je často nutné zajistit prioritizaci příslušných datových toků, aby nedošlo k výraznému zhoršení kvality hovorů. Typicky je možné využít mechanismů a nástrojů, které na základě přednastavených parametrů klasifikují provoz tak, aby obsluha paketových front byla prováděna na základě této klasifikace.

Koncept SDN přináší možnost centralizovaně ovládat či konfigurovat síťové prvky v reálném čase na základě dat poskytovaných aplikací třetí strany. V tomto smyslu by mělo být možné nakonfigurovat SIP signalizační server (např. Asterisk nebo Kamailio) tak, aby na základě údajů ze signalizačních zpráv předával SDN kontroleru síťové parametry probíhajících RTP streamů. SDN kontroler by reagoval dynamickým nastavováním QoS pravidel pro danou síť a tím zlepšil kvalitu probíhajících hovorů, které jsou ze své podstaty citlivé na vytíženost sítě. Tato metodika fundamentálně zbavuje síťové prvky odpovědnosti za klasifikaci, která nebude závislá na potencionálních nestandardních vlastnostech RTP streamů, jako např. šifrování.

Cílem této práce je navrhnout metodiku dynamické prioritizace RTP streamů na SDN prvcích na základě informací poskytovaných protokolem SIP. Tato metodika je implementována ve zvoleném SIP serveru a nasazena do testovacího prostředí, kde je otestována na scénáři, ve kterém by za normálních okolností proběhlo zhoršení kvalitativních parametrů hovoru.

Před samotným návrhem jsou popsány dílčí využité koncepty a protokoly. V prvních dvou kapitolách jsou popsány koncepty QoS a SDN. Další dvě kapitoly jsou dedikovány protokolu SIP a SIP serveru Kamailio, který se ukázal vhodným pro účely této práce.

Kapitola 2

Quality of Service

Díky konvergenci sítí se nyní provozují prakticky všechny typy služeb na paketové bázi na totožných fyzických prostředcích. Tím se sice snížila provozní cena, ale u některých služeb, zejména telefonních, to může být na úkor kvality přenosu, jelikož sdílejí přenosové médium s jinými typy provozu. Z tohoto důvodu může být vhodné zavést prioritizaci těch služeb, které jsou citlivé na přenosové parametry jako zpoždění a jitter, čímž rozumíme tzv. QoS (Quality of Service). Konvergovanou síť si lze představit jako cestu, po které cestují dopravní prostředky bez ohledu na jejich účel, ovšem to se může jevit jako problémové, když cestou musí projet sanitka. Jedním možným řešením by bylo rozšířit cestu, což by ovšem popíralo základní účel konvergence, tedy snížení ceny. Druhým řešením je zavedení pravidel na křižovatkách mezi dopravními úseky tak, aby sanitky měly vždy přednost před ostatními vozidly.

QoS nepředstavuje žádnou konkrétní službu nebo protokol, je to spíše koncept, který stojí za některými síťovými prvky a nástroji. Může být rozdělen na 2 základní komponenty:

- Lokální operace - aplikace QoS nástrojů na konkrétním směrovači.

- Zdrojová signalizace - značení paketů tak, aby se každý síťový prvek na trase jednoznačně a konzistentně rozhodnul, jaké QoS nástroje aplikovat.

Toto může být přirovnáno k IP směrování a přepojování (forwarding). Směrovací informace je platná pro všechny směrovače v síti a její koncept je mnohem komplexnější oproti přepojování, které je implementováno individuálně a nezávisle na ostatních prvcích. Zasadním rozdílem u zmíněných QoS termínů je absence standardizovaných specifikací, které jsou v networkingu běžné. Z toho plyne, že neexistuje jednotná odpověď pro to, jak by měly být směrovače konfigurovány pro dosažení QoS chování mezi koncovými klienty.

2.1 Standardy

Dva hlavní standardy v IP sítích relevantní ke QoS jsou **IntServ** a **DiffServ**. IntServ je popsán v RFC1663 a DiffServ v RFC2475.

IntServ byl vyvinut jako protokol rezervující prostředky na end-to-end bázi pro jednotlivé datové toky, ovšem kvůli své komplexnosti nebyl nikdy běžně nasazován. Některé jeho koncepty však byly převzaty v MPLS sítích, konkrétně protokolem RSVP.

Model DiffServ je vyvinut na bázi tříd. Provoz je klasifikován do tříd služeb spíše než do toků, jako je prováděno v IntServ. Další zásadní rozdíl je absence end-to-end signalizace, jelikož v DiffServ modelu každý router operuje nezávisle.

V DiffServ modelu router rozlišuje mezi různými typy provozu aplikací klasifikačního procesu. Jakmile je provedena klasifikace, jsou na každý specifický typ provozu aplikovány QoS nástroje pro docílení požadovaného chování. Nicméně platí, že pravidla klasifikačního procesu a jejich vztah k rozhodování o tom, které QoS nástroje jsou aplikovány na který typ provozu, jsou definovány lokálně na každém routeru. Tento elementární koncept se nazývá per-hop chování (PHB).

V PHB není přítomna žádná signalizace mezi sousedy nebo koncovými body a QoS chování je na každém routeru definováno lokální konfigurací. Z tohoto vyvstávají dvě zřejmé otázky. Zaprvé, jak dosáhnout koherence v chování provozu napříč routery a zadruhé, jak propagovat informace mezi routery.

Koherence je dosažena konzistentní konfigurací, která zajišťuje, že klasifikační proces na každém routeru, jímž provoz prochází, dochází ke stejným závěrům při rozhodování, které QoS nástroje budou aplikovány na jaký typ provozu. Naneštěstí má koncept PHB Achillovu patu. End-to-end chování QoS politik v síti může být kompromitováno, jestliže byť jen jeden router v cestě datového toku neaplikuje stejná QoS pravidla. Konzistence v tomto kontextu však neznamená, že by všechny routery musely mít identické konfigurace.

Druhá výzva kladená PHB konceptem je otázka, jak sdílet informace mezi routery bez mechanismu zajišťujícího signalizaci mezi nimi či koncovými body. První úloha při odeslání nebo přijetí paketu routerem je klasifikace. Pokud chce router, který posílá paket, sdělit signalizační informaci sousednímu routeru, jedinou možností je změnit obsah paketu za využití QoS nástrojů. Úprava paketu může takto ovlivnit klasifikační proces na routeru, jež daný paket obdrží. Úspěch této operace ovšem závisí na konfiguraci obdržujícího routeru, který při jeho klasifikačním procesu nemusí zohlednit předanou informaci.

2.2 Charakterizace provozu

Jak bylo řečeno, různé typy provozu vyžadují odlišný přístup. Typickými parametry, které zohledňujeme při charakterizaci provozu, jsou zpoždění, jitter a ztrátovost.

- **Zpoždění** neboli latence je definováno jako doba mezi vysláním paketu ze zdroje a přijetím stejného paketu na cílovém bodě. Matematicky by se dalo vyjádřit jako rozdíl mezi časem odeslání a přijetí. Typicky je značeno Δ a měřeno v milisekundách.

- **Jitter** reprezentuje variaci ve zpoždění mezi po sobě jdoucími pakety. Tedy pokud má paket 1 zpoždění Δ_1 a paket 2 zpoždění Δ_2, jitter mezi pakety 1 a 2 se bude rovnat rozdílu zpoždění $\Delta_1 - \Delta_2$.

- **Ztrátovost** reprezentuje podíl paketů, které nebyly nebyly přijaty koncovým bodem ku celkovému počtu odeslaných paketů. Typicky je zapisována v procentech.

Z hlediska senzitivity na tyto 3 parametry je důležité rozlišovat mezi provozem real-time a provozem nonreal-time. Tato práce se zaměřuje zejména na IP telefonní provoz, spadající do kategorie real-time, která je citlivá zejména na zpoždění a jitter. Je možno tvrdit, že tyto parametry jsou z hlediska QoS pro daný typ provozu relevatní.

Zpoždění je důležité, jelikož real-time pakety jsou pro adresáta relevantní pouze pokud jsou obdrženy v časovém úseku, ve kterém jsou očekávány. Přiliš vysoké zpoždění může paket znehodnotit. Různé míry zpoždění ovlivňují kvalitu hovoru. Účastníci typicky zaznamenají nesrovnalosti při zpoždění nad 250 ms a doporučení ITU-T G.114 udává maximální zpoždění 150 ms. Tyto hodnoty ovšem platí pro celou datovou cestu, která může zahrnovat Internet - zpoždění vznikající v lokální síti by proto mělo být podstatně nižší.

Jitter může způsobovat inkonzistence v hlasu a vyšší hodnoty mohou výrazně ovlivnit srozumitelnost. Koncové aplikace pro eliminaci jitteru typicky využívají buffer, který pozdrží přijaté pakety tak, aby byly rozdíly v časech jejich zpracování konstantní (v ideálním případě). Jitter tedy lze vykompenzovat umělým vkládáním zpoždění. Tyto buffery jsou typicky efektivní pro variace ve zpoždění do 100 ms.

[1]

2.3 Obsluha paketových front

Stěžejními QoS nástroji pro řízení stavů zahlcení sítě jsou právě metody obsluhy paketových front. V průběhu let jich vznikla řada. Základní modely front jsou:

- Základní řazení FIFO (First In First Out)

- Vlastní řazení do front CQ (Custom Queuing)

- Vážené řazení do front WFQ (Weighted Fair Queuing)

- Prioritní řazení do front PQ (Priority Queuing)

Kombinací základních modelů lze získat složitější modely front. Nejčastěji je možné se setkat s těmito:

- Vážené řazení do front na základě třídy CBWFQ (Class-Based Weighted Fair Queuing)
- Prioritní řazení do front / Vážené řazení do front PQ/WFQ (Priority Queuing / Weighted Fair Queuing)
- Řazení do front s nízkou latencí LLQ (Low Latency Queuing)

2.3.1 FIFO

Jak napovídá název, prvním paketem na výstupu této fronty je vždy paket, který byl ze všech uchovávaných paketů nejdříve na vstupu. Všechny pakety jsou tedy obsluhovány jedinou frontou jednotným způsobem. Pro většinu výrobců síťových prvků je toto výchozí využívaný model.

2.3.2 WFQ

WFQ bývá označováno jako "kruhový výběr po bitech", jelikož implementuje mechanismus, jehož obsluha není založena na paketech, ale na bitech. Z toho plyne, že si je vědomo velikosti paketů a větší pakety nebudou oproti těm menším zvýhodněny. Datový tok je dle klasifikace rozdělen do front, kterým je alokována poměrná část přenosového pásma rozhraní. Počet front je tedy proměnlivý. V případě, že dojde k vyprázdnění libovolné fronty, je její alokované pásmo dynamicky rozděleno mezi ostatní fronty.

2.3.3 CQ

Metoda CQ taktéž pracuje na rozdělení toku do více obslužných front, ovšem přidělené přenosové pásmo může být nastaveno na konkrétní hodnotu nebo na procentuální podíl z celkové dostupné šířky pásma. Stejně jako v případě metody WFQ je nevyužité pásmo dostupné i ostatním frontám.

2.3.4 PQ

Aplikace citlivé na přenosové parametry, jako je zpoždění nebo ztrátovost, lze obsloužit prioritně. V případě, že je provoz takto klasifikován a zařazen do fronty o vyšší prioritě, bude tato fronta obsluhována bez přerušení až do jejího vyprázdnění. Fronty mohou být rozděleny mezi více prioritních úrovní, kdy pak obecně platí, že fronta o vyšší prioritě má vždy přednost před frontou o nižší prioritě. Ačkoli je zde zmiňována vyšší priorita, v praxi je typicky reprezentována relativně nižším číslem.

Nevýhodou tohoto přístupu může být přílišné omezení ostatního provozu při soustavném zaplňování prioritní fronty.

2.3.5 CBWFQ

CBWFQ rozšiřuje metodu WFQ o uživatelem definované parametry. Třídy provozu jsou definovány na základě protokolů, seznamů ACL či vstupních rozhraní, a rozděleny do front typu FIFO.

Každé třídě mohou být přiřazeny parametry jako např. šířka pásma, váha a paketový limit. Šířka pásma přiřazená třídě je garantovaná při zahlcení rozhraní (congestion). Po překročení limitů fronty může dojít k zahazování paketů náležících dané frontě.

2.3.6 PQ/WFQ

V tomto modelu je prioritizována fronta PQ. Kombinací s frontou WFQ je zajištěna rafinovanější obsluha i pro méně prioritní typy provozu. Z hlediska reálného provozu je PQ/WFQ výhodnější než model PQ.

2.3.7 LLQ

LLQ přináší prioritní obsluhu PQ do CBWFQ. Do nastavené prioritní fronty je doporučováno směrovat pouze telefonní provoz, přičemž ostatní provoz obsluhovat na bázi CBWFQ. Na rozdíl od LQ se zde u prioritní fronty definuje maximální přenosová rychlost tak, aby nedošlo ke znemožnění přenosu na ostatních frontách.

[1][2]

Kapitola 3

Software Defined Networks

Řídícím softwarem rozumíme inteligenci/algoritmus stanovující optimální cesty a reagující na výpadky a nové požadavky sítě. Ve své podstatě je SDN o přenesení tohoto řídícího softwaru ze síťových zařízení do centrální výpočetní jednotky, která má přehled o celé síti.

Dnešní síťová zařízení jsou navržena s ohledem na jejich nezávislost a schopnost přizpůsobit se neustálým změnám v síti, což ovšem - jak přibývaly jejich nové funkce, nároky a implementace - v průběhu času vedlo k soustavně narůstající komplexitě. Jeden z hlavních motivů pro vytvoření konceptu SDN sítí bylo právě zjednodušení těchto zařízení. Vhodnou analogií k této změně přístupu by mohl být přechod z architektur procesorů CISC na RISC, kdy se komplexita přesunula ze samotné architektury procesorů na programy, které je využívají. SDN navíc kromě zjednodušení zamotných zařízení přináší příležitost zjednodušit i jejich správu a nahradit takřka primitivní nástroje jako SNMP a CLI.

Pomocí SDN můžeme identifikovat provoz jednotlivých aplikací a nakládat s nimi různými způsoby, přestože jsou směrovány na stejný koncový bod. Mnoho specializovaných síťových zařízení jako firewally, load balancery nebo IDS (Intrusion Detection System) spoléhá na hlubší inspekci paketů. Kontrolou adres a portů jsou tato zařízení schopna oddělit a izolovat provoz. Vezmeme-li v úvahu, že směrovací tabulky budou centralizovatelně programovatelné kontrolerem se znalostí topologie sítě, mohla by být funkce takovýchto zařízení integrována přímo do přepínačů.

SDN se pokouší rozdělit síťové procesy následujícím způsobem:

- **Přepojování (forwarding), filtrování, prioritizace**
  Odpovědnost za expedici implementovanou v hardwarových tabulkách zůstává na zařízení. Stejně tak jsou lokálně prováděny funkce, jako filtrování založené na ACL, či prioritizace provozu.

- **Řízení**
  Komplikovaný řídící software je odstraněn ze zařízení a umístěn na centralizovaný kontroler, který má úplný přehled o síti a schopnost provádět optimální expediční a směrovací rozhodnutí.

Základní expediční hardware na síťovém zařízení je možné programovat extérním softwarem kontroleru.

- **Aplikace**
  Nad kontrolerem běží aplikační procesy implementující funkce vyšších vrstev a účastnící se při rozhodování o tom, jak zpracovávat a řídit expedici a distribuci paketů v síti.

3.1 SDN zařízení

SDN zařízení se skládá z API pro komunikaci s kontrolerem, abstrakční vrstvy a funkce zpracování paketů. V případě fyzického přepínače představuje funkci zpracování paketů příslušný hardware, zatímco u virtuálního přepínače je tato funkce zastoupena softwarem. Abstrakční vrstva představuje jednu nebo více flow tabulek. Logika zpracování paketů se skládá z mechanismů založených na výsledcích zpracování příchozích paketů a nalezení shody nejvyšší priority. Když je nalezena shoda, příchozí paket je zpracován lokálně, pokud není explicitně přeposlán na kontroler. V případě, že shoda nalezena není, duplikát paketu může být poslán na kontroler pro další zpracování.

3.1.1 Flow tabulky

Flow tabulky jsou základní datové struktury v SDN zařízení. Tyto tabulky umožňují zařízením posoudit příchozí pakety a podstoupit odpovídající akci v závislosti na obsahu paketu. Tyto akce mohou zahrnovat přeposlání na konkrétní porty, zahození paketu, přeposlání paketu na všechny porty (flooding), apod.

Flow tabulky se skládají z jednotlivých záznamů, které typicky tvoří dvě komponenty, *match fields* a *akce*. Match fields jsou využívány pro porovnání vůči příchozím paketům. Příchozí paket je porovnán vůči match field dle prioritního uspořádání a je vybrána první úplná shoda. Akce jsou instrukce, které by mělo síťové zařízení provést, pokud nalezne shodu paketu s match field v příslušném záznamu.

3.1.2 Virtuální SDN přepínače

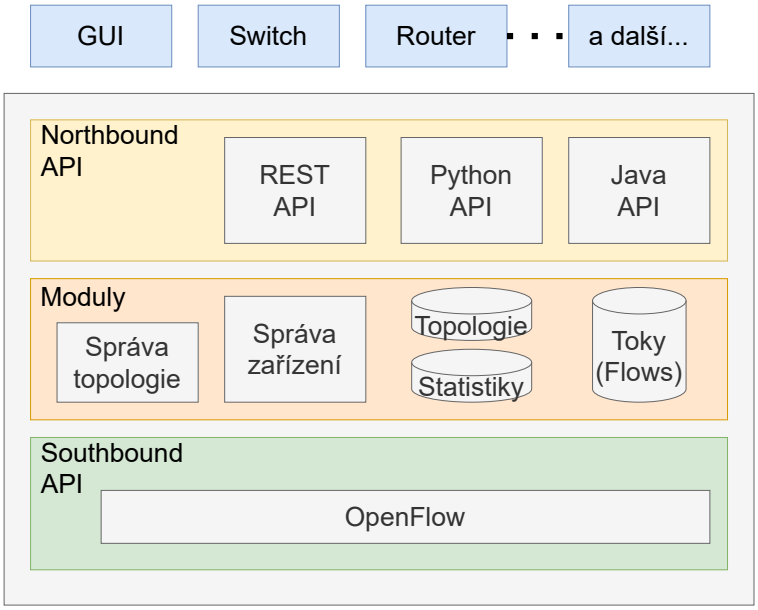
Implementace SDN zařízení v softwaru je nejjednodušší způsob vytvoření SDN zařízení, jelikož flow tabulky, jejich záznamy a match fields budou jednoduše namapovány na obecné datové struktury jako třízená pole a hashovací tabulky. V důsledku je pravděpodobnější, že se dvě virtuální SDN zařízení vytvořené různými vývojovými týmy budou chovat konzistentněji než v případě dvou různých hardwarových implementací. Naopak, implementace v softwaru bude pravděpodobněji pomalejší a méně efektivní než ta implementovaná v hardwaru.

Virtuální implementace také méně trpí nedostatkem zdrojů jako výpočetní výkon nebo velikost paměti. Z toho plyne, že zatímco hardwarová SDN implementace bude podporovat relativně

limitované množství flow záznamů, u softwarové implementace může být tento limit řádově vyšší. Typickým nástrojem pro virtualizaci SDN sítě, který bude využíván i zde, je **MiniNet**.

3.2 SDN kontroler

Bylo řečeno, že kontroler udržuje přehled o celé síti, implementuje pravidla, řídí všechna zařízení tvořící síťovou infrasturkturu, a poskytuje northbound API. Kontrolery často přichází s vlastní sadou aplikačních modulů jako je přepínač, směrovač, jednoduchý firewall a load balancer.



Obrázek 3.1: Anatomie SDN kontroleru

3.2.1 Základní moduly SDN kontroleru

Kontroler poskytuje detaily SDN controller-device protokolu tak, aby byly výše zmíněné aplikace schopny komunikovat s SDN zařízeními jejich hlubší znalosti. Obrázek 3.1 vyobrazuje Northbound API komunikující s aplikacemi a Southbound API, jež představuje OpenFlow komunikující s SDN zařízeními. Mezi těmito porty leží základní funkcionality poskytované každým SDN kontrolerem. Mezi ně patří:

- **Lokace koncových zařízení:** laptopy, stolní počítače, tiskárny, mobilní zařízení, atd.
- **Lokace síťových zařízení:** přepínače, směrovače, bezdrátová AP
- **Řízení síťové topologie** - Udržuje informace o spojeních mezi síťovými zařízeními a o připojených koncových zařízeních.

- **Řízení toků** - Udržuje databázi datových toků řízených kontrolerem a provádí nezbytnou koordinaci s zařízeními, zajišťujíce synchronizaci jejich toků s řečenou databází.

[3]

3.2.2 Ryu

Ryu je open-source SDN kontroler napsaný v Pythonu. Podporuje různé protokoly pro správu síťových zařízení, jako je OpenFlow, Netconf, OF-config, atd. V rámci OpenFlow plně podporuje verze 1.0 až 1.5 a rozšíření Nicira.

Hlavní kód kontroleru je organizován ve složce */ryu/*. Následuje výčet jeho klíčových komponentů:

- **app/** - Obsahuje sadu aplikací pracujících nad kontrolerem.
- **base/** - Obsahuje základní třídu pro Ryu aplikace. Třída RyuApp v *app_manager.py* je děděna při vytváření nové aplikace.
- **lib/** - Obsahuje sadu paketových knihoven pro zpracovávání hlaviček různých protokolů a knihovny pro OFConfig. Zahrnuje i parsery pro NetFlow a sFlow.
- **ofproto/** - Obsahuje informace specifické OpenFlow protokolu a příslušné parsery pro podporu jeho různých verzí.
- **topology/** - Obsahuje kód provádějící mapování topologie a zpracování příslušných informací o OpenFlow přepínačích (porty, spoje, apod). Interně využívá protokol LLDP.

3.2.2.1 Možnosti QoS konfigurace

Ryu umožňuje řadu konfiguračních úkonů pomocí HTTP zpráv směrovaných na jeho IP adresu. Metoda zprávy definuje povahu úkonu.

- **GET** - vyžádání konfiguračních položek nacházejících se na dané URL
- **POST** - vložení nových konfiguračních položek
- **DELETE** - vymazání položek, které odpovídají parametrům zprávy

Tyto zprávy jsou zasílány na URL, které specifikuje typ konfigurační položky a konkrétní přepínač. Pro práci s nastavením front jsou dostupné následující URL:

- **/qos/queue/status/{switch_id}** - Metodou GET lze na tomto URL vyžádat status nastavených frontách.

- **/qos/queue/{switch_id}** - Zvolenou metodou se nastaví, smaže nebo vyžádá konfigurace front. Metodou POST se v tomto případě vždy smaže předchozí platná konfigurace.

- **/qos/rules/{switch_id}** - Na této URL lze nastavit pravidla, na základě kterých se bude přepínač rozhodovat, do které fronty zařadit konkrétní datový tok. Přidaná pravidla jsou definována obsahem zprávy POST, který může specifikovat např. IP adresy nebo port. V případě úspěšného přidání pravidla kontroler pošle v odpovědi jeho identifikační číslo, dle kterého je možné dané pravidlo smazat zprávou metody DELETE.

Namísto parametru `switch_id` lze doplnit "all"pro aplikaci na všechny přepínače náležící kontroleru. [4]

3.3 OpenFlow

SDN definuje jak komunikační protokol mezi SDN datovou vrstvou a SDN řídící vrstvou, tak část chování datové vrstvy. Nepopisuje chování samotného kontroleru.

OpenFlow systém se skládá z OpenFlow kontroleru, který komunikuje s jedním nebo více OpenFlow přepínači. OpenFlow protokol definuje specifické zprávy a formát zpráv vyměňující se mezi kontrolerem (řídící vrstva) a zařízením (datová vrstva). Openflow specifikuje, jak by zařízení měla reagovat v různých situacích a jak by měla odpovídat na příkazy kontroleru.

OpenFlow je protokolová specifikace, která popisuje komunikaci mezi OpenFlow přepínači a OpenFlow kontrolerem. Následují základní operace OpenFlow řešení:

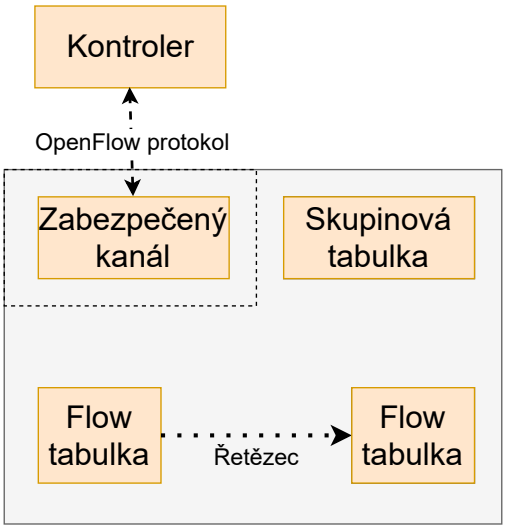
- Kontroler zaplní tabulky směrovače.

- Přepínač posuzuje hlavičky příchozích paketů, hledá odpovídající záznamy a provede příslušnou akci. S ohledem na situaci je zpracována hlavička druhé vrstvy a poté potenciálně třetí a čtvrté vrstvy.

- Pokud odpovídající záznam nalezen není, přepínač paket přepošle na kontroler, aby se rozhodl, jak s paketem naložit.

- Je-li přijat takový paket, kontroler typicky doplní přepínač o nové záznamy tak, aby další pakety mohl zpracovat lokálně. Je možné, že kontroler nastaví tzv. wildcard pravidla, která se vztahují na více datových toků zároveň.

Nejnovější dostupná verze OpenFlow je 1.5, ovšem většina současných implementací je z důvodu sjednocení postavena na verzi 1.3. Následující popis OpenFlow tedy bude vztažen právě k verzi 1.3.



Obrázek 3.2: Dílčí části OpenFlow přepínače

3.3.1 OpenFlow přepínač

OpenFlow přepínač se skládá z jedné nebo více flow tabulek a skupinové tabulky, která provádí inspekci a expedici, a OpenFlow kanálu pro komunikaci s kontrolerem. Přepínač komunikuje s kontrolerem a kontroler řídí přepínač přes OpenFlow protokol. Za využití OpenFlow protokolu může kontroler přidat, aktualizovat či smazat flow záznamy ve flow tabulkách, ať už reaktivně nebo proaktivně. Každá flow tabulka v přepínači obsahuje sadu flow záznamů a každý takový záznam se skládá z match polí, počítadel a sady instrukcí.

Hledání shody začíná na první flow tabulce a může pokračovat k dalším flow tabulkám. Mezi flow záznamy je hledán shodný záznam dle priority, přičemž je použita první shoda v dané tabulce. Pokud je nalezena shoda, jsou vykonány instrukce příslušící danému záznamu. Pokud ve flow tabulce žádná shoda nalezena není, následující akce závisí na "table-miss"záznamu: paket může být například přeposlán na kontroler přes OpenFlow kanál, zahozen nebo se přesměrován na další flow tabulku.

Instrukce přidružené ke každému flow záznamu buď obsahují akci nebo modifikují řetězcové zpracování (pipeline processing). Akce zahrnuté v instrukcích popisují expedici paketů, modifikaci paketů a zpracovávání dle skupinových tabulek. Instrukce řetězcového zpracování umožňují vyslání paketů na další tabulky a výměnu informací mezi tabulkami ve formě metadat. Řetězové zpracování se zastaví ve chvíli, kdy instrukční sada příslušící flow záznamu ve shodě nespecifikuje další tabulku. v tomto případě je paket většinou modifikován a expedován.

Flow záznamy mohou směrovat na porty čímž jsou typicky myšleny fyzické porty, ale může se také jednat o "logické"porty definované přepínačem nebo rezervované porty definované specifikací OpenFlow. Rezervované porty mohou specifikovat generické směrovací akce, jako je posílání na kontroler, flooding nebo expedice za užití metod mimo OpenFlow specifikaci.

Akce asociované s flow záznamy mohou také nasměrovat pakety na skupinu, která specifikuje dodatečné zpracování. Skupiny reprezentují sady akcí pro flooding a také komplexnější směrovací sémantiku (např. vícecestné směrování, rychlé přesměrování, agregace linek). Skupiny také umožňují vícero flow záznamům směrovat na společný identifikátor. Tato abstrakce umožňuje efektivně měnit společné výstupní akce napříč flow záznamy.

Skupinová tabulka obsahuje skupinové záznamy; každý skupinový záznam obsahuje seznam souborů akcí (action buckets) se specifickou sémantikou závislou na skupinovém typu. Akce v jednom nebo více souborech akcí jsou aplikovány na pakety spadající do skupiny.

3.3.2 OpenFlow porty

Tato kapitola popisuje abstrakci OpenFlow porty a různé typy OpenFlow portů podporovaných OpenFlow.

OpenFlow porty jsou síťová rozhraní, přes která procházejí pakety mezi OpenFlow zpracováním a zbytkem sítě. OpenFlow přepínače jsou mezi sebou propojeny přes OpenFlow porty.

OpenFlow přepínač tvoří množství OpenFlow portů dostupných přes OpenFlow procesy. Sada OpenFlow portů nemusí být totožná se sadou síťových rozhraní přepínače - některá rozhraní mohou být pro OpenFlow zakázaná a OpenFlow přepínač může definovat dodatečné OpenFlow porty.

OpenFlow pakety jsou přijímány na **vstupním portu** (ingress port) a zpracovány OpenFlow řetězcem, který je může přepojit na **výstupní port** (output port). Vstupní port paketu náleží propusti OpenFlow řetězce a reprezentuje OpenFlow port, na kterém byl paket přijat. Vstupní port může být použit při hledání odpovídající instrukce. OpenFlow řetězec se může rozhodnout pro vyslání paketu na výstupní port za využití výstupní akce, která definuje, jak je paket vyslán zpět do sítě.

OpenFlow přepínač musí podporovat 3 typy OpenFlow portů: *fyzické porty*, *logické porty* a *rezervované porty*.

Standardní porty OpenFlow standardní porty jsou definovány jako fyzické porty, logické porty a lokální rezervované porty (pokud jsou podporovány). Standardní porty mohou být použity jako vstupní a výstupní porty, použity ve skupinách a mohou mít čítače portů.

Fyzické porty Openflow fyzické porty jsou přepínačem definované porty, které korespondují s hardwarovými rozhraními přepínače. V případech, kdy je OpenFlow přepínač virtualizován, OpenFlow fyzický port může reprezentovat virtuální protějšek hardwarového rozhraní.

Logické porty Openflow fyzické porty jsou přepínačem definované porty, které přímo nekorespondují s hardwarovými rozhraními přepínače. Logické porty jsou abstrakce vyšší úrovně, které mohou být definovány metodami nenáležícími do OpenFlow (např. agregované linky, tunely, loopback roz-

hraní). Logické porty mohou zahrnovat enkapsulaci paketů a mohou být namapovány na různé fyzické porty.

Jediný rozdíl mezi fyzickými a logickými porty je, že paket asociovaný s logickým portem může mít dodatečné (meta)datové pole zvané `Tunnel-ID`. Když je paket přijat na logickém portu a vyslán na kontroler, logický i přidružený fyzický port jsou nahlášeny kontroleru.

Rezervované porty OpenFlow rezervované porty jsou definovány jejich specifikací. Specifikují generické přepojovací akce jako posílání dat na kontroler, flooding nebo přepojování za užití metod mimo OpenFlow specifikaci.

Od přepínače je vyžadována podpora určitých rezervovaných portů, které zde budou popsány.

- `ALL:` Reperezentuje všechny porty, které přepínač může použít pro přepínání specifického paketu. Může být použit pouze jako výstupní port.
- `CONTROLLER:` Reprezentuje kontrolní kanál s OpenFlow kontrolerem. Může být použit jako vstupní i výstupní port.
- `TABLE:` Reprezentuje začátek OpenFlow řetězce. Tento port je platný pouze ve výstupní akci v seznamu akcí zprávy *packet-out* a předá paket ke zpracování první tabulkou v OpenFlow řetězci.
- `IN_PORT:` Reprezentuje vstupní port paketu. Může být použit pouze jako výstupní port, tedy pro vyslání paketu stejným portem, na kterém byl přijat.
- `ANY:` Speciální hodnota používaná v některých OpenFlow příkazech, pokud port není specifikován (port wildcarded). Nemůže být použit jako vstupní ani výstupní port.

Dále přepínač **může** podporovat porty `LOCAL`, `NORMAL` a `FLOOD`.

3.3.3 OpenFlow tabulky

3.3.3.1 Řetězcové zpracování

Přepínače s podporou OpenFlow mohou být buď *OpenFlow-only* nebo *OpenFlow-hybrid*. **OpenFlow-only** přepínače podporují pouze OpenFlow operace - v těchto přepínačích jsou všechny pakety zpracovávány OpenFlow řetězcem a nemohou být zpracovány jinak.

OpenFlow-hybrid přepínače podporují jak OpenFlow operace, tak normální ethernetové operace jako L2 přepínání, VLAN, L3 směrování, ACL a QoS. Tyto přepínače by měly poskytovat klasifikační mechanismus mimo OpenFlow, který směruje provoz buď do OpenFlow řetězce nebo do normálního řetězce.

OpenFlow řetězec (pipeline) každého OpenFlow přepínače obsahuje alespoň jednu flow tabulku obsahující flow záznamy. OpenFlow řetězcové zpracování definuje, jak pakety interagují s těmito tabulkami.

3.3.3.2 Flow tabulka

Flow tabulka se skládá z flow záznamů. Každý záznam obsahuje:

- match pole: porovnáváním paketu se mezi těmito poli hledá shoda. Skládá se ze vstupního portu, hlaviček paketu a volitelně metadat specifikovaných předešlou tabulkou.
- priorita: specifikuje možnou přednost před ostatními záznamy v dané tabulce
- čítače: pro aktualizaci paketů ve shodě
- instrukce: modifikuje sadu akcí nebo zpracovávající řetězec
- timeout: maximální doba platnosti daného flow záznamu
- cookie: dodatečná data daná kontrolerem. Mohou být použita kontrolerem k vyfiltrování statistik toků nebo jejich modifikaci a smazání - není používáno při zpracovávání paketu.

Záznam flow tabulky je identifikován match poli a prioritou, tj. kombinace match pole a priority je pro každý záznam v tabulce unikátní. Záznam, který použije wildcard na všechna pole (bude platit pro jakýkoli paket) a jeho priorita je 0, se nazývá table-miss flow záznam.

3.3.4 OpenFlow kanál

OpenFlow kanál je rozhraní, které spojuje OpenFlow přepínač a kontroler. Skrze toto rozhraní kontroler konfiguruje a spravuje přepínač, přijímá zprávy z přepínače a expeduje z něj pakety. Mezi datovou cestou a OpenFlow kanálem se rozhraní liší dle implementace, ovšem všechny zprávy přenášené přes OpenFlow kanál musí být formátovány dle OpenFlow protokolu. OpenFlow kanál je typicky šifrovaný za využití TLS, ale může může operovat i přes samotný TCP protokol.

OpenFlow protokol podporuje 3 typy zpráv, "*controller-to-switch*", *asynchronní* a *symetrické*, a různé jejich podtypy. Controller-to-switch zprávy jsou inicializovány kontrolerem a používány k přímé správě nebo inspekci stavu přepínače. Asynchronní zprávy jsou inicializovány přepínačem a slouží k informování kontroleru o síťových událostech a změnách ve stavu přepínače. Symetrické zprávy mohou být inicializovány jak přepínačem, tak kontrolerem.

[5]

Kapitola 4

SIP

Je to textově založený řídící protokol aplikační vrstvy, který byl navržen a vytvořen skupinou IETF a spadá pod kategorii VoIP protokolů. Nejdůležitější RFC dokument pro SIP je RFC 3261, který specifikuje jádro protokolu. [6]

Za zmínku stojí i to, že SIP je založený na protokolu HTTP, tudíž mezi nimi lze nalézt podobnost v hlavičkách a kódových označeních odpovědí.

4.1 Význam SIP

Jak již název napovídá (Session Initation Protocol), jedná se o protokol, jehož účelem je vytváření, modifikace a ukončování relace s jedním či více účastníky. Relací rozumíme soubor odesílatelů, příjemců a stav udržovaný mezi těmito odesílateli a příjemci během jejich komunikace. Příkladem takových relací mohou být internetové telefonní hovory, konference, streamování apod. Jsou zde ale i další protokoly, které se na komunikaci podílí. SIP typicky spolupracuje s protokoly RTP a SDP. RTP se používá pro přenos médií, stará se o kódování a dělení dat do paketů. Jeho zabezpečená verze je SRTP. SDP je textově založený protokol sloužící pro popis a vyjednání parametrů spojení, jako např. použité kodeky a typy médií. [7]

SIP podporuje 5 základních aspektů vytváření a ukončování komunikace:

- **Lokace uživatele** - určení koncového systému, který pro komunikaci bude použit
- **Dostupnost uživatele** - určení, zdali je volaná strana ochotna komunikovat
- **Schopnosti uživatele** - určení médií a jejich parametrů, které se budou používat
- **Nastavení relace** - vyzvánění (ringing) a navázání relačních parametrů na volané i volající straně

- **Relační management** - přenos a ukončení relací, modifikace jejich parametrů a vyvolávání služeb

[6]

4.2 SIP URI

SIP účastníci jsou indentifikováni za pomoci SIP URI, která má následující formu: `sip:jméno@doména`, například `sip:bob@vsb.cz`. Skládá se z uživatelského jména, pod kterým je účastník zaregistrován, a doménového jména serveru, na kterém je uživatelské jméno zaregistrováno. Velmi podobně jsou řešeny emailové adresy. Doménové jméno lze zaměnit za IP adresu serveru a celé SIP URI lze zaměnit za IP adresu adresáta, ovšem SIP URI je ze své podstaty snáze zapamatovatelné.

4.3 Prvky SIP architektury

Přestože by si v jednoduché topologii vystačili dva účastníci sami, běžná topologie se skládá z více SIP entit. Základní z nich jsou user agenti, proxy, registrar a redirect servery. Tyto entity jsou převážně logického charakteru - jedno zařízení často zastává více těchto entit, záleží však na konkrétní implementaci.

4.3.1 User agent

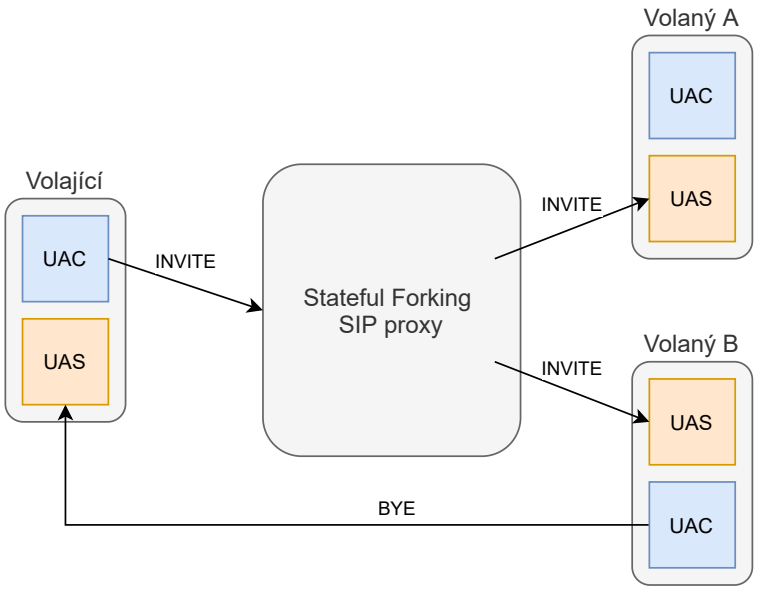
Koncové body telekomunikační sítě, které jako signalizační protokol pro sestavení spojení používají SIP, nazýváme *User Agent* (UA). V dnešní době může UA představovat široká škála zařízení a programů. User agent je často označován i jako *User Agent Server* (UAS) a *User Agent Client* (UAC). UAS i UAC jsou ovšem pouze logickými entitami, každý User agent je zároveň UAC i UAS.

UAC je část user agenta, která posílá žádosti a přijímá odpovědi. Oproti tomu UAS je část, která přijímá žádosti a posílá odpovědi, viz obr. 4.1 . Jelikož user agent představuje zároveň UAS i UAC, lze o něm dle situace říci, že se chová jako UAS nebo UAC.

B2BUA je speciální typ user agenta, který zastává funkci SIP proxy serveru, ale chová se jako UA. Typickým příkladem je **Asterisk**. Oproti SIP proxy může poskytovat řadu pokročilejších doplňkových služeb a proto je využíván jako pobočková ústředna pro řádově až tisíce uživatelů. SIP proxy je ovšem využitelná pro řádově vyšší počet uživatelů.

4.3.2 SIP proxy

Pro účely této práce se nebude využívat serveru typu B2BUA, ale klasického SIP proxy - konkrétně **Kamailia**. Předpokládá se, že žádaná konfigurace bude probíhat čistě na základě jednotlivých SIP zpráv a nebude nutné využívat žádné doplňkové služby nad médii.



Obrázek 4.1: UAC a UAS

SIP proxy jsou důležité prvky SIP infrastuktury, které primárně směrují SIP zprávy, ale mohou poskytovat i mnoho dalších služeb jako například registraci, autentizaci atd. Vzhledem ke konfiguraci se rozlišuje mezi 2 typy SIP proxy serveru – stateful a stateless (stavové a bezestavové).

4.3.2.1 Stateless

Tyto SIP proxy pracují pouze jako jednoduché směrovače, které SIP zprávy směrují nezávisle na předchozích. Přestože jsou tyto zprávy obvykle uspořádány do transakcí, stateless proxy na to neberou ohled. Díky tomu jsou řádově rychlejší než stateful proxy, ale na druhou stranu však nejsou schopny vykonávat pokročilejší operace jako větvení nebo přesměrování, nezachytí replikaci zpráv a pomaleji detekují nekonečné smyčky.

4.3.2.2 Stateful

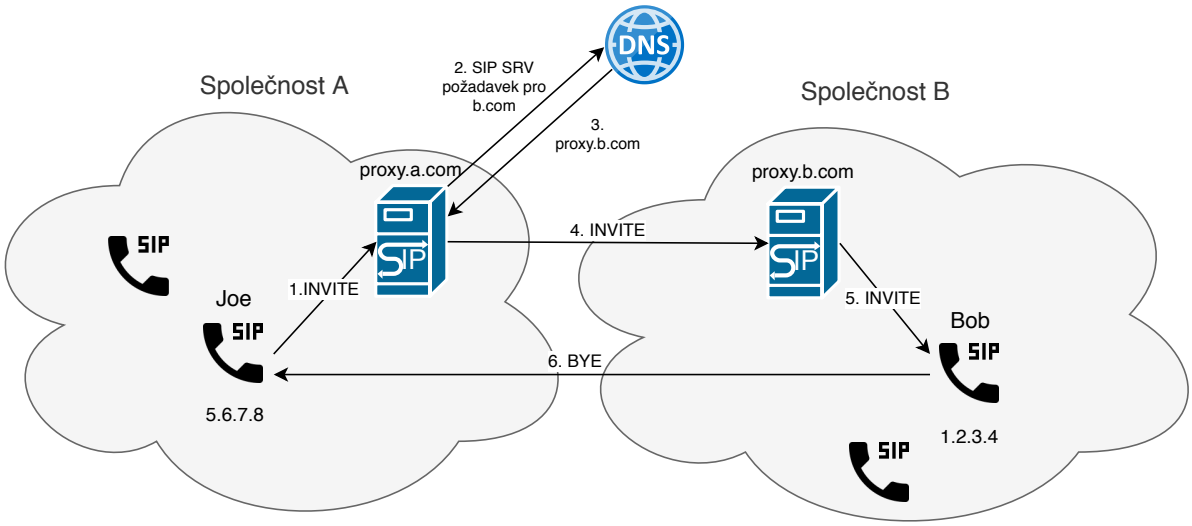
Tyto SIP proxy jsou komplexnější. Po přijetí požadavku server vytvoří záznam, který uchovává až do ukončení transakce nebo dialogu. Dle toho se dělí na **transakční** a **dialogové**. Některé transakce mohou trvat poměrně dlouho, obzvláště ty vytvořené metodou INVITE, což výrazně snižuje výkon serveru. Schopnost asociovat si jednotlivé zprávy do transakcí propůjčuje stateful proxy zajímavé vlastnosti.

- **Větvení** - na základě přijetí jedné zprávy může server vyslat 2 a více zpráv, viz. obr 4.1 .
- **Zachycení opakovaných zpráv** - z uchovávaného záznamu může zjistit, jestli už byla stejná zpráva přijata.

- **Lokalizace uživatele** - může provádět komplikovanější metody pro nalezení uživatele; například v situaci ve které volaný nezvedá telefon v kanceláři, je hovor přesměrován na jeho mobilní telefon.

Většina dnešních SIP proxy jsou stateful a často podporují generování záznamů o spojeních, větvení a některé typy NAT.

Bežně má každá centrálně administrovaná entita, například firma, vlastní SIP proxy server, využívaný všemi UA v dané entitě. Předpokládejme, že máme společnosti A a B a obě mají svůj vlastní proxy server. Obrázek 4.2 ukazuje, jak se zpráva pro zahájení relace od zaměstnance Joe z firmy A dostane k zaměstnanci Bob z firmy B. Joe použije URI adresu `sip:bob@b.com`. Joeův UA je nakonfigurován tak, že všechny výchozí žádosti posílá na `proxy.a.com` (zná IP adresu). Tento proxy server z URI adresy `sip:bob@b.com` pozná, že se volaný nachází v jiné firmě, proto pokud nemá v paměti patričný záznam, přeloží si přes DNS server adresu `proxy.b.com` na příslušnou IP adresu, kam posléze zprávu přepošle. Proxy server `proxy.b.com` ví, že Bob právě sedí ve své kanceláři a je dostupný přes telefon na svém stole, který má IP adresu `1.2.3.4`, na kterou zprávu pošle. Odpověd již není třeba posílat přes SIP proxy, jelikož v žádosti byla obsažena fyzická adresa odesílatele.



Obrázek 4.2: SIP topologie

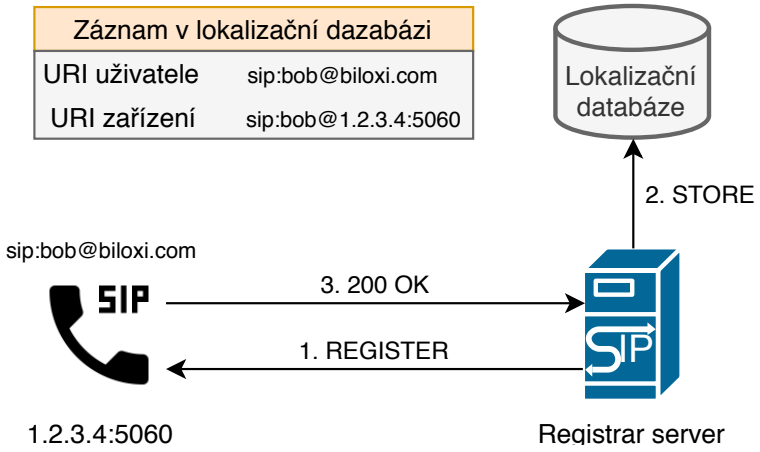
4.3.3 Registrar

V předchozím příkladu bylo zmíněno, že SIP proxy server `proxy.b.com` sice zná Bobovu aktuální polohu, ale aby měl tuto informaci, musí být jeho UA registrován na SIP registrar serveru. Registrar server je speciální entita, která od uživatelů přijímá registrace. Tímto získává informace o jejich aktuální poloze (v tomto případě IP adresa, port a uživatelské jméno), jež si ukládá

do lokalizační databáze (location database). Účel lokalizační databáze je v tomto případě přeložení adresy `sip:bob@b.com` na `sip:bob@1.2.3.4:5060`. Obrázek 4.3 znázorňuje typickou SIP registraci. UA na registrar pošle zprávu metody REGISTER, která obsahuje adresu záznamu User URI `sip:bob@biloxi.com` a Device URI `sip:bob@1.2.3.4:5060`, kde adresa `1.2.3.4` je IP adresa telefonu. Registrar tuto informaci přečte a uloží ji do lokalizační databáze. Pokud vše proběhlo v pořádku, registrar pošle telefonu odpověď 200 OK a proces registrace je hotov.

Každá registrace má omezenou životnost, hlavička `Expires` nebo parametr `expires` v hlavičce `Contact` udává, po jakou dobu je registrace platná. UA proto musí registraci obnovovat, v opačném případě platnost registrace vyprší.

[7]



Obrázek 4.3: SIP registrace

4.3.4 Redirect server

Entita, která přijímá požadavky a posílá zpět odpovědi obsahující seznam současných poloh daného uživatele je nazývána *redirect server*. Redirect server přijme požadavek a vyhledá si daného adresáta v příslušné lokalizační databázi vytvořené registrar serverem. Poté vytvoří seznam současných lokací a pošle jej zpět tvůrci žádosti v odpovědi třídy 3xx. Toto může být užitečné například v situaci, kdy se osoba zrovna nachází v doméně `cvut.cz` a nabízí se jí možnost využívat jejich SIP proxy. Běžně ovšem využívá SIP proxy v doméně `vsb.cz`, takže by ji bez redirect serveru nebylo možné kontaktovat pomocí původní SIP URI.

[2]

4.4 SIP zprávy

SIP signalizace se skládá ze série zpráv, které jsou buď žádosti nebo odpovědi, a tyto zprávy jsou nezávisle posílány přes danou síť. Typicky jsou neseny v jednotlivých UDP datagramech, ačkoli je

možné použít i TCP protokol. Každá SIP zpráva se skládá z "prvního řádku", hlavičky a těla, které je definováno SDP protokolem. Přenos těchto SIP zpráv mezi jednotlivými SIP prvky znázorňuje tzv. SIP trapezoid, viz. obr. 4.4

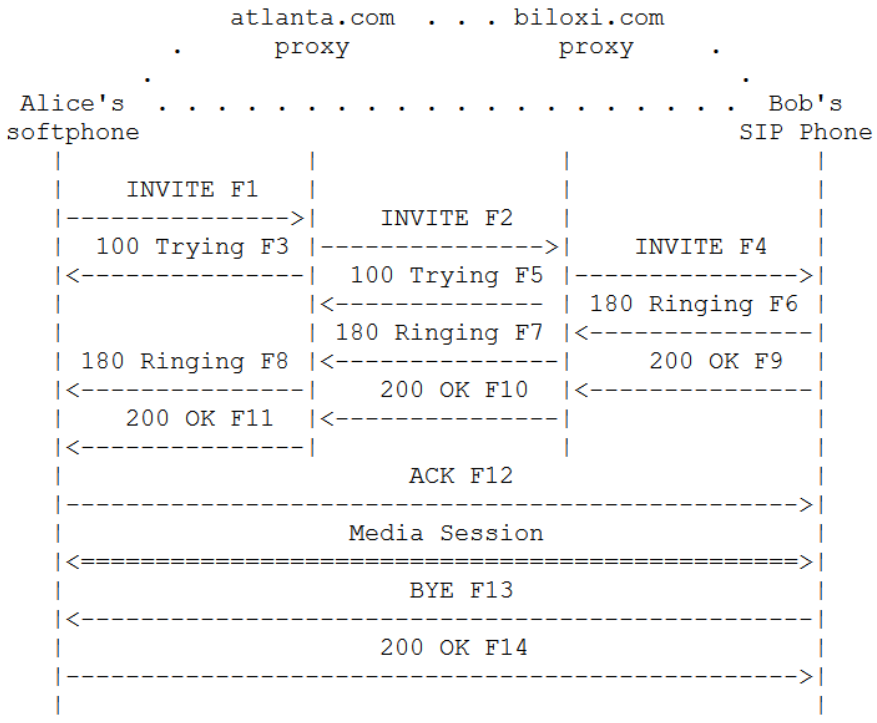
Struktura zprávy je následující:

- **První řádek** (start-line) - liší se u požadavku a odpovědi
  - **Řádek požadavku** (request-line) - při požadavku
    - **Metoda**
    - **SIP URI**
    - **Verze SIP** - typicky 2.0
  - **Řádek statusu** (status-line) - při odpovědi
    - **Verze SIP**
    - **Statusový kód** (status-code) - třímístné číslo značící povahu odpovědi
    - **Důvod** (reason-phrase) - oproti statusovému kódu je vyjádřen textem a určen pro lidského účastníka
- **Hlavička** (message-header)
  - **Via** - každé zařízení v rámci požadavku připíše svůj "Via"řádek, do kterého zapíše svou fyzickou adresu a identifikátor transakce (branch). U odpovědi je zpráva posílána stejnou cestou zpět a tyto řádky se odebírají
  - **Max-Forwards** - udává, kolik skoků může zpráva do cíle provést (mezi SIP síťovými prvky)
  - **To** - obsahuje zobrazované jméno (display name) a SIP URI, na kterou je zpráva směřována
  - **From** - obsahuje zobrazované jméno a SIP URI odesílatele zprávy, taktéž obsahuje parametr tag obsahující náhodnou sérii znaků pro identifikační účely
  - **Call-ID** - obsahuje globálně unikátní identifikátor pro daný hovor generovaný náhodnou kombinací znaků a jménem koncového zařízení nebo IP adresou; kombinace parametrů To tag, From tag, a Call-ID kompletně definuje peer-to-peer vztah mezi účastníky a je označován jako dialog
  - **Cseq** (Command sequence) - obsahuje číslo, které je inkrementováno každým dalším požadavkem, a název metody
  - **Contact** - obsahuje SIP URI, které reprezentuje přímou cestu pro kontaktování odesílatele a je bežně složené z plně specifikovaného doménového jména (FQDN), ale je možno použít i IP adresu

  - **Content-Type** - uvádí jakého typu (povahy) je tělo zprávy
  - **Content-Lenght** - uvádí, kolik bytů má tělo zprávy

- **Prázdný řádek** (CRLF)

- **Tělo** (message-body) - SDP. Obsahuje parametry popisující relaci a média. Zde jsou popsány ty nejdůležitější.
  - **o** - identifikuje relaci a účastníka, který ji vytvořil
  - **c** - obsahuje informace o spojení, např. IP adresa, na které klient poslouchá
  - **m** - popis médií včetně využívaných portů
  - **k** - šifrovací klíč



Obrázek 4.4: SIP trapezoid

Příklad SIP zprávy `F1` v trapezoidu 4.4 :

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
```

```
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

(Zde následuje SDP)

[6]

Kapitola 5

SIP proxy Kamailio

Kamailio vzniklo v červnu 2005 rozdělením projektu SIP Express Routeru (SER). Nově započatý projekt měl za cíl vytvořit open-source vývojové prostředí pro stavbu robustního, škálovatelného open-source SIP serveru. Původní název byl OpenSER, ale z důvodu porušení ochranné známky byl v červenci roku 2008 název projektu změněn na Kamailio. V listopadu roku 2008 se vývojářské týmy Kamailia a SERu znovu spojily a stejně tak konvergovaly i jejich projekty Kamailiem verze 3.0.0. V této práci se využívá jedné z nejnovějších verzí 5.4.

5.1 Architektura

Kamailio má modulární architekturu. Zjednodušeně jeho komponenty můžeme rozdělit do dvou kategorií:

- **Jádro** - komponenta, která poskytuje nízkoúrovňové funkcionality. Počínajíc verzí 3.0.0, jádro Kamailia obsahuje takzvané interní knihovny. Tyto knihovny sbírají kód sdílený několika moduly, u kterých není důvod, aby byly součástí jádra.
- **Moduly** - komponenty, které poskytují většinu funkcionalit

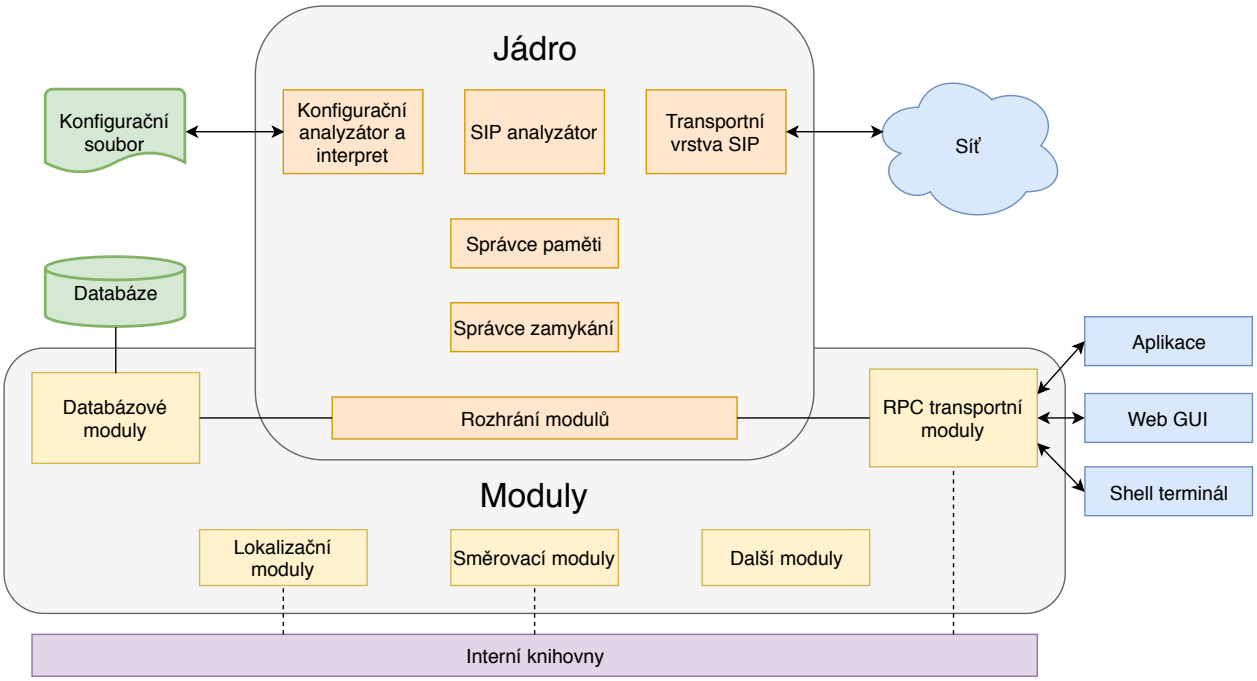
V obrázku 5.1 je přibližně znázorněna současná architektura Kamailia. [8]

5.2 Konfigurační soubor

Konfigurační soubor Kamailia využívá svůj vlastní skriptovací jazyk, jehož struktura je plně popsána v dokumentaci [9]. Zde budou kromě samotného konfiguračního souboru popsány i některé základní prvky tohoto jazyka.

5.2.1 Struktura

Struktura konfiguračního souboru `kamailio.cfg` se dělí na 3 části:



Obrázek 5.1: Architektura Kamailia

- Globální parametry (global parameters)
- Nastavení modulů (modules settings)
- Směrovací bloky (routing blocks)

Je doporučeno je udržet v tomto pořadí, ale není to ve všech případech nutné.

5.2.2 Generické prvky

5.2.2.1 Komentáře

```
#toto je nejcasteji pouzivany radkovy komentar
//taktez radkovy komentar
/*blokovy
komentar*/
```

5.2.2.2 Hodnoty

Existují 3 typy:

- integer - čísla reprezentována 32-bitovou hodnotou

- boolean - logická 1 (true, on, yes) nebo logická 0 (false, off, no)
- string - bloky textu ohraničené uvozovkami ("...", '...')

5.2.2.3 Identifikátory

Nejsou ohraničeny uvozovkami a řídí se pravidly pro integer a boolean hodnoty. Jsou to například parametry a funkce jádra, funkce modulů, atd., například:

```
return
```

5.2.2.4 Proměnné

Začínají znakem $, například:

```
\$var(x) = $rU + "@" + $fd;
```

5.2.2.5 Akce

Jsou to prvky užívané uvnitř směrovacích bloků zakončené znakem ";". Mohou vykonávat funkci z jádra nebo modulu, podmínku, smyčku, nebo přiřazovací výraz.

```
sl_send_reply("404", "Not found");
exit;
```

5.2.2.6 Výrazy

Jsou to skupiny tvrzení, proměnných, funkcí, a operátorů

```
if(!t_relay())

if($var(x)>10)

"sip:" + $var(prefix) + $rU + "@" + $rd
```

5.2.3 Směrovací bloky

Směrovací bloky se na rozdíl od ostatních částí konfiguračního souboru zpracovávají "za běhu". Při přirovnání k běžným programovacím jazykům je můžeme vnímat jako funkce. Směrovací blok je identifikován specifickým označením následovaným jménem v hranatých závorkách. Následují akce ve složených závorkách. Jméno může být tvořeno jakýmkoli alfanumerickým stringem, jehož text nemusí být nutně ohraničen uvozovkami.

```
route_block_id[NAME] {
   ACTIONS
}
```

Následující bloky mohou být vykonávány na základě síťových událostí (např. přijetí SIP zprávy), časovačů nebo událostí specifických pro moduly. Dále se mohou vyskytovat i tzv. subsměrovací bloky, které jsou, obdobně jako funkce, vyvolané ze směrovacích bloků. Struktura je zřejmá z následujícího příkladu.

```
request_route{
   ...
   route("test");
   ...
}

route["test"]{
   ...
}
```

request_route Tento směrovací blok je vykonáván pro každou SIP žádost – může být považován za ekvivalentní k funkci "main()".

route Takto se označují výše uvedené subsměrovací bloky. Mohou být "zavolány"z kteréhokoli bloku (dříve pouze z bloku "request_route"), ale je zásadní do nich vkládat akce validní pro kořenový blok. Jejich formát je totožný s běžnými směrovacími bloky s tím rozdílem, že vrací integer tomu bloku, který je zavolal (např. "return 0"). Tuto hodnotu lze uchovat přes `$rc` proměnné.

Vrácená hodnota subsměrovacího bloku je vyhodnocována následujícím způsobem:

- záporná hodnota vrací logickou 0 (false)
- 0 interpretuje `exit`
- kladná hodnota vrací logickou 1 (true)

reply_route Obsahuje akce, které budou vykonány pro každou SIP odpověď, kterou Kamailio obdrží ze sítě. Dle RFC 3261 se v tomto bloku nesmí provádět žádné akce vykonávající směrování, ale může být použita akce `drop`. Tento blok je volitelný.

onreply_route Toto je směrovací blok vykonáván modulem `tm`. Obsahuje akce, které se vykonávají v kontextu aktivních transakcí.

onsend_route Je vykonáván pouze při předávání požadavků (forwarding). Mohou zde být kontrolovány parametry pro cílovou destinaci (IP adresy, porty, protokoly, apod.).

event_route Generický typ směrovacího bloku vykonávaný při specifických událostech. Jeho struktura je následující: `event_route[groupid:eventid]`.

- groupid - mělo by být stejné jako jméno směrovacího bloku, ze kterého se vykoná
- eventid - volitelný smysluplný text popisující událost

[9]

5.3 Nástroje

5.3.1 kamctl

Tento nástroj je obsažen v základní instalaci. Jeho konfigurační soubor `kamctlrc` je umístěn ve stejné složce jako konfigurační soubor `kamailio.cfg`. Je potřeba jej upravit nastavením SIP domény a databázového enginu, ale většina ostatních parametrů má své výchozí hodnoty. Používá se k řízení Kamailia z příkazového řádku a poskytuje širokou škálu možných operací a výpisů.

5.3.2 kamdbctl

Je obsažen v základní instalaci. Využívá stejnou konfiguraci jako `kamctl`. Může být využit k vytvoření a správě databázové struktury vyžadované Kamailiem. Toto by měla být jedna z prvních procedur po instalaci Kamailia. Nepoužívá se však ke správě záznamů databázových tabulek, nýbrž pouze ke správě databázové struktury a přístupu k ní.

5.3.3 kamcmd

Je obsažena v základní instalaci. `Kamcmd` je aplikace, která je Kamailiu schopna posílat RPC příkazy z příkazového řádku. Vyžaduje, aby byl v Kamailiu načten `ctl` modul.

[10] [11]

Kapitola 6

Návrh řešení

Jak bylo popsáno v kapitole o konceptu QoS, fundamentálním problémem standardu IntServ je zajištění signalizace mezi koncovými body. Z toho důvodu se typicky přikláníme ke standardu DiffServ, který ovšem zpravidla spoléhá na statickou konfiguraci klasifikace provozu. Když se podíváme na právě ten typ provozu, pro který je QoS nejrelevantnější - tedy IP telefonní provoz - zjistíme, že využívá signalizaci, jež by se pro konfiguraci QoS parametrů dala příhodně využít. SIP v naprosté většině využívá služeb SIP serveru, který má přehled o probíhajících hovorech, ale v klasických IP sítích nemá prostředky k ovlivnění konfigurace síťových prvků.

Koncept SDN přináší možnost komunikovat s extérními aplikacemi skrze centrální řídící prvek - SDN kontroler. SIP server by takto měl být schopen informovat SDN kontroler o probíhajících hovorech a ten by příslušné RTP streamy prioritizoval před ostatním provozem přímo na síťových prvcích. Na základě této myšlenky se pokusím navrhnout funkční systém, který bude názorně nastavovat QoS parametry sítě na základě informací vyměňovaných protokolem SIP.

Žádaný systém by měl plnit následující funkce:

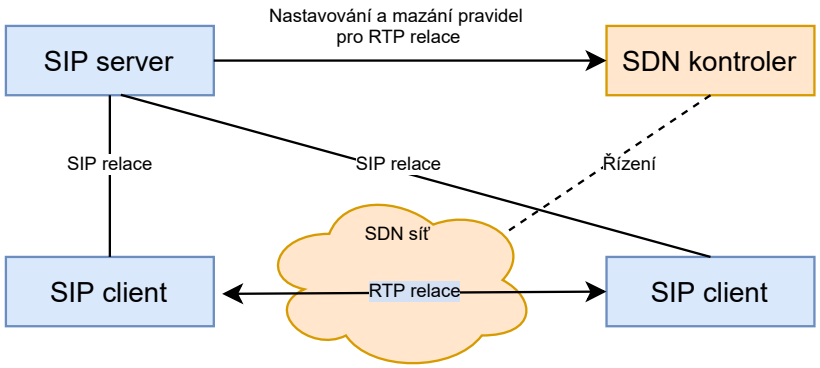
1. Jakmile je hovor zahájen, SIP server předá SDN kontroleru síťové parametry příslušných RTP streamů - ty jsou obsaženy v SDP požadavcích INVITE a odpovědí 200 OK.

2. SDN kontroler na základě přijatých informací nastaví prioritizaci daných RTP streamů na síťových prvcích.

3. Po ukončení hovoru, tedy za přijetí požadavku BYE, SIP server informuje SDN kontroler a ten vyřadí pravidla nastavená pro daný hovor.

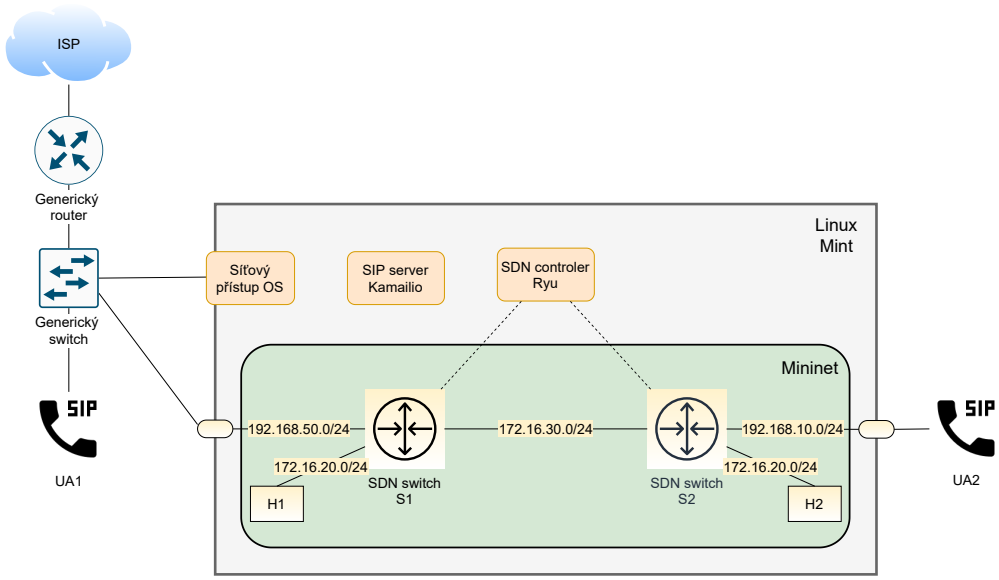
6.1 Pracovní topologie

S ohledem na technické možnosti, povahu využívaných technologií a zachování jednoduchosti řešení, je SDN síť realizována virtuálně na Linuxovém systému za využití síťového emulátoru **Mininet**.



Obrázek 6.1: SIP-SDN-QoS interakce

Aby bylo možné znatelně a měřitelně ovlivňovat QoS parametry na daných síťových uzlech, SDN síť se bude kromě SDN kontroleru skládat ze dvou virtuálních SDN přepínačů, mezi nimiž se bude mísit RTP provoz mezi telefonními klienty (UA1 a UA2) s provozem mezi virtuálními hosty (H1 a H2). Při aplikaci QoS pravidel bude žádoucí prioritizovat provoz mezi UA klienty před provozem mezi virtuálními hosty.



Obrázek 6.2: Pracovní topologie

6.2 Výměna informací mezi SIP serverem a SDN kontrolerem

Jako SIP server jsem si zvolil SIP proxy Kamailio. Síťové parametry lze z SDP těla SIP zpráv vyčíst pomocí modulu *sdpops*[12]. Nejpříhodnějším prostředkem k předávání informací kontroleru je využití protokolu HTTP. Kamailio může pro posílání a zpracovávání HTTP zpráv vy-

užít moduly *http_async_client*[13] a *http_client*[14]. Modul *http_client* pracuje oproti modulu *http_async_client* synchronně, což znamená, že po vyslání HTTP požadavku čeká na obdržení odpovědí a až poté pokračuje v procesu, ze kterého byla funkce zavolána. Z hlediska efektivity by se dalo usoudit, že bude vždy lepší použít modul *http_async_client*, nicméně je třeba počítat s těmito vlastnostmi:

- **http_async_client** neumí předávat proměnné zpět do procesu, ze kterého byla jeho funkce zavolána
- **http_client** umí posílat pouze HTTP požadavky metod PUT a POST

SDN kontroler si parametry uloží pod inkrementujícím se identifikačním číslem, které pošle zpět na SIP server v HTTP odpovědi 200 OK. Toto číslo posléze bude sloužit ke smazání záznamu. Zde vyvstává problém, že Kamailio typicky uchovává stavy, tedy i proměnné, v rámci transakce. Transakce, v rámci které se daná pravidla vytvoří, končí přijetím hovoru. K překonání tohoto problému poslouží modul *dialog*[15], jenž za přijetí požadavku INVITE zaznamená probíhající dialog, ke kterému lze přidružit libovolné proměnné.

Kamailio za přijetí požadavku BYE vyšle na kontroler HTTP zprávu metody DELETE obsahující zaznamenané identifikační číslo, pro což využije modulu *http_async_client*.

Kapitola 7

Realizace řešení

Ze začátku byly pomocí příkazu *apt* nainstalovány následující balíčky. Patří mezi ně zejména síťový emulátor Mininet, nástroje pro další instalace jako jsou *git* a *make*, a nezbytné knihovny.

```
# apt update
# apt install git mininet gcc g++ flex bison libmysqlclient-dev make
    autoconf pkg-config libssl-dev libcurl4-openssl-dev libxml2-dev libpcre3-
    dev mysql-server libevent-dev
```

7.1 Kamailio

Jelikož je potřeba do Kamailia přidat některé nestandardní moduly, byla zvolena instalace z Gitu, ve které lze tyto moduly zahrnout do kompilace. Do konfiguračního souboru kompilačního systému Kamailia se tedy zahrnou již řečené moduly *http_client*, *http_async_client* a modul zvolené databáze - v našem případě *db_mysql*.

```
# git clone --depth 1 --no-single-branch https://github.com/kamailio/
    kamailio kamailio
# cd kamailio
# make cfg include_modules="db_mysql http_client http_async_client"
# make all
# make install
```

Konfigurační soubory Kamailia jsou v tomto typu instalace umístěné jinde než v případě instalace z balíčků, tj. `/usr/local/etc/kamailio/`. V konfiguračním souboru `kamctlrc` byly pouze odkomentovány následující řádky a zadána SIP doména. Řádek s parametrem `DBRWPW` byl odkomentován, aby se nemuselo zadávat heslo při zápisu a čtení z databáze (např. tvorba a výpis účtů).

```
SIP_DOMAIN = localhost
```

```
DBENGINE = MYSQL
DBRWPW="kamailiorw"
```

Pro výsledné otestování systému za účasti dvou klientů byla vytvořena databáze a dva účty.

```
# cd /usr/local/etc/kamailio/
# kamdbctl create
# kamctl add 100 100
# kamctl add 200 200
```

7.1.1 kamailio.cfg

Na začátku konfiguračního souboru je potřeba definovat typ databáze. Tento kód aktivuje již existující konfigurační bloky.

```
#!define WITH_MYSQL
```

Na konec sekce načítání modulů se přidají řečené moduly.

```
loadmodule "http_async_client.so"
loadmodule "dialog.so"
loadmodule "http_client.so"
loadmodule "sdpops.so"
```

Úvodní konfigurací v hlavním konfiguračním bloku se dle specifikace modulu nastaví uchovávání dialogových stavů. K takovému stavu lze přiřadit další data, např. proměnné, a tak vyměňovat informace mezi jednotlivými transakcemi. Dialog je automaticky zahozen po přijetí "BYE"nebo vypršení časového limitu `default_timeout`, který je resetován za každého přijetí požadavku příslušného dialogu.

```
modparam("dialog", "default_timeout", 100)
...
request_route {
   ...
   if(is_method("INVITE") && !has_totag())
   {
      $dlg_ctx(timeout_route) = "DLGTIMEOUT";
      $dlg_ctx(timeout_bye) = 1;
   }
   dlg_manage();
   ...
```

Po přijetí požadavku metody INVITE se z SDP těla zprávy uloží do proměnných typu *dlg_var* IP adresa a port, které si klient určil pro RTP relaci. Ty se využijí až v momentě, kdy bude hovor přijat.

```
request_route {
   ...
   if (is_method("INVITE")) {
      sdp_get_line_startswith("$avp(mline)", "m=audio");
      $dlg_var(invport)=$(avp(mline){s.select,1, });

      sdp_get_line_startswith("$avp(cline)", "c=IN IP4");
      $dlg_var(invip)=$(avp(cline){s.select,2, });
   }
   ...
}
```

Za přijetí požadavku BYE vyšle Kamailio pomocí asynchronní funkce `http_async_query` na SDN kontroler HTTP zprávu metody DELETE (dle dokumentace modulu Dialog je proměnná typu `$dlg_var` dostupná až po provedení `loose_route()`[15]). Tělo takové zprávy bude obsahovat identifikační číslo pravidla obdržené v odpovědi 200 OK při jeho vytvoření.

```
route[WITHINDLG] {
   ...
   if (loose_route()) {
      if (is_method("BYE")){
         $http_req(all) = $null;
         $http_req(timeout) = 100;
         $http_req(method) = "DELETE";
         $http_req(hdr) = "testHeader: header";
         $http_req(body) = '{"qos_id": "'+$dlg_var(qos_id)+'"}';
         $http_req(suspend) = 0;
         http_async_query("http://localhost:8080/qos/rules/all", "OVS_POST
             ");

         $http_req(all) = $null;
         $http_req(timeout) = 100;
         $http_req(method) = "DELETE";
         $http_req(hdr) = "testHeader: header";
         $http_req(body) = '{"qos_id": "'+$dlg_var(qos_id2)+'"}';
         $http_req(suspend) = 0;
```

```
            http_async_query("http://localhost:8080/qos/rules/all", "OVS_POST
                ");
        }
        ...
    }
    ...
}
```

Stejně jako u požadavku INVITE se zpracuje SDP tělo odpovědi 200 OK na požadavek INVITE. Paremetr *$rm*, který je součástí podmínky, obsahuje metodu v poli *CSeq*, jež odpovídá metodě požadavku iniciujícího transakci.[6] V této fázi dialogu se hovor může považovat za zahájený a pro UA1 a UA2 se vyšlou IP adresy a porty vyextrahované z požadavku INVITE v HTTP zprávě metody POST na SDN kontroler. Funkce *http_client_query* je synchronní, tedy počká na odpověď SDN kontroleru a do dialogové proměnné si z ní uloží identifikační číslo nastaveného pravidla.

```
reply_route {
    ...
    if (status=="200" && $rm=="INVITE"){
        ### UA1
        http_client_query("http://localhost:8080/qos/rules/all", '{"match":
            {"nw_dst": "' + $dlg_var(invip) + '", "nw_proto": "UDP", "tp_dst
            ": "' + $dlg_var(invport) + '"}, "actions":{"queue": "1"}}', "
            $dlg_var(qos_id)");
        $dlg_var(qos_id)=$(dlg_var(qos_id){s.select,5,,});
        $dlg_var(qos_id)=$(dlg_var(qos_id){s.numeric});

        ### UA2
        sdp_get_line_startswith("$avp(mline)", "m=audio");
        $var(port)=$(avp(mline){s.select,1, });

        sdp_get_line_startswith("$avp(cline)", "c=IN IP4");
        $var(ip)=$(avp(cline){s.select,2, });

        http_client_query("http://localhost:8080/qos/rules/all", '{"match":
            {"nw_dst": "' + $var(ip) + '", "nw_proto": "UDP", "tp_dst": "' +
            $var(port) + '"}, "actions":{"queue": "1"}}', "$dlg_var(qos_id2)"
            );

        $dlg_var(qos_id2)=$(dlg_var(qos_id2){s.select,5,,});
```

```
        $dlg_var(qos_id2)=$(dlg_var(qos_id2){s.numeric});
    }
    ...
}
```

Funkci *http_async_query* může náležet konfigurační blok, který se provede již nezávisle na hlavním procesu Kamailia. Zde je možné si nechat vypsat např. chybové kódy HTTP odpovědi, ovšem již nelze předávat proměnné zpět do hlavního procesu, což je zásadní rozdíl oproti funkci *http_client_query*.

```
route[OVS_POST] {
    xlog("L_INFO", "route[OVS_POST]: INIT\n");
    if ($http_ok) {
        xlog("L_INFO", "body and length: $http_rb\n $http_bs\n");
    } else {
        xlog("L_INFO", "route[OVS_POST]: error $http_err)\n");
    }
}
```

7.2 Konfigurace SDN prvků

SDN kontroler Ryu lze nainstalovat i z balíčků, ale následující instalací z GitHubu za využití nástroje `pip-requires` lze předejít některým problémům s verzemi knihoven.

```
# git clone https://github.com/faucetsdn/ryu.git
# sed '/OFPFlowMod(/,/)/s/0, cmd/1, cmd/' ryu/ryu/app/rest_router.py > ryu
    /ryu/app/qos_rest_router.py
# pip3 install ryu/tools/pip-requires
# python3 ryu/setup.py install
```

Spustí se MiniNet v základní konfiguraci se dvěma Open vSwitch přepínači a externím kontrolerem. K oběma přepínačům jsou připojeni hosté *h1* a *h2*.

```
# mn --topo linear,2 --mac --switch ovsk --controller remote -x
```

V Mininetu lze k jednotlivým entitám přistupovat podobně jako k linuxovým terminálům. Za název entity lze například zadat příkazy pro smazání a přidání IP adresy nebo výchozí brány. Výchozí brány jsou v našem případě IP adresy, které později budou přidány na SDN přepínače. Rozhraní a propojení jednotlivých entit si lze vypsat příkazem `net`.

```
h1 ip addr del 10.0.0.1/8 dev h1-eth0
h1 ip addr add 172.16.20.10/24 dev h1-eth0
```

```
h2 ip addr del 10.0.0.2/8 dev h2-eth0
h2 ip addr add 172.16.10.10/24 dev h2-eth0

h1 ip route add default via 172.16.20.1
h2 ip route add default via 172.16.10.1
```

Ze systémového terminálu lze přímo ovlivňovat některé parametry jednotlivých SDN přepínačů, zejména definování verze OpenFlow nebo propojení s fyzickým rozhraním na zařízení. v tomto případě bude rozhraní `enx00e04c02e0a8` vést k telefonnímu klientovi UA1 a rozhraní `enx9cebe82ef7ac` ke klientovi UA2.

```
# ovs-vsctl set Bridge s1 protocols=OpenFlow13
# ovs-vsctl set Bridge s2 protocols=OpenFlow13
# ovs-vsctl set-manager ptcp:6632
# ovs-vsctl add-port s1 enx00e04c02e0a8
# ovs-vsctl add-port s2 enx9cebe82ef7ac
```

V samostatném terminálu se spustí aplikace Ryu kontroleru společně s API, která budou v tomto řešení potřeba. Po zadání tohoto příkazu se spustí terminál kontroleru a připojí se k němu vytvořené OvS přepínače.

```
# ryu-manager ryu.app.rest_qos ryu.app.qos_rest_router ryu.app.
    rest_conf_switch
```

Jakmile se přepínače připojí ke kontroleru, lze k nim přistupovat skrze nasazené API. Pro přidání záznamu do SDN přepínače se nástrojem *curl* vytvoří HTTP zpráva metody POST směrovaná na URL, jež reprezentuje flow tabulku konkrétního přepínače. v tomto případě pracují SDN přepínače v režimu směrovače, proto do nich lze vložit IP adresy a výchozí brány. Oproti klasickým sítím se při vkládání IP adresy nespecifikuje rozhraní.

```
# curl -X POST -d '{"address": "172.16.20.1/24"}' http://localhost:8080/
    router/0000000000000001
# curl -X POST -d '{"address": "172.16.30.10/24"}' http://localhost:8080/
    router/0000000000000001
# curl -X POST -d '{"address": "192.168.50.51/24"}' http://localhost:8080/
    router/0000000000000001
# curl -X POST -d '{"gateway": "172.16.30.1"}' http://localhost:8080/
    router/0000000000000001

# curl -X POST -d '{"address": "172.16.10.1/24"}' http://localhost:8080/
    router/0000000000000002
```

```
# curl -X POST -d '{"address": "172.16.30.1/24"}' http://localhost:8080/
    router/0000000000000002
# curl -X POST -d '{"gateway": "172.16.30.10"}' http://localhost:8080/
    router/0000000000000002
# curl -X POST -d '{"address": "192.168.10.1/24"}' http://localhost:8080/
    router/0000000000000002
```

Zprávou PUT se nastaví přístup do OVSDB (The Open vSwitch Database Management Protocol), bez kterého by nebylo možné přistupovat k nastavení front. Následující zprávou POST se nastaví dvě fronty – jedna výchozí a druhá s minimální propustností, ke které se budou vázat RTP streamy zjištěné na Kamailiu. V případě, že není specifikováno rozhraní, jako v tomto případě, je určené pravidlo platné pro všechna nasazená rozhraní. Narozdíl od předchozích konfiguračních položek je konfigurací fronty vždy přepsána jejich poslední platná konfigurace.

```
# curl -X PUT -d '"tcp:127.0.0.1:6632"' http://localhost:8080/v1.0/conf/
    switches/0000000000000001/ovsdb_addr
# curl -X PUT -d '"tcp:127.0.0.1:6632"' http://localhost:8080/v1.0/conf/
    switches/0000000000000002/ovsdb_addr
# curl -X POST -d '{"type": "linux-htb", "max_rate": "200000", "queues
    ":[{"max_rate": "200000"}, {"min_rate": "100000"}]}' http://localhost
    :8080/qos/queue/all
```

Nastavené fronty by samy o sobě měly fungovat na bázi obsluhové metody CBWFQ. Fronty využívají uživatelem definované parametry, jako je třeba garantované přenosové pásmo, a více datových toků vyskytujících se v jedné z nastavených front by mělo automaticky obdržet poměrnou část dostupného pásma. Na úrovni nastavování klasifikace provozu, v tomto případě v konfiguraci Kamailia, lze v záznamu přiřazující konkrétní datový tok definovat i prioritu, která by do systému přidala funkci prioritní obsluhy PQ.

[4]

Kapitola 8

Testování a analýza

Nasazený systém by měl být v tomto bodě schopen navazovat hovory pomocí SIP protokolu a na SDN přepínačích dynamicky nastavovat pravidla prioritizující existující RTP relace. V této kapitole bude demonstrována funkcionalita tohoto systému a výsledný efekt na QoS parametry v jednoduchém scénáři.

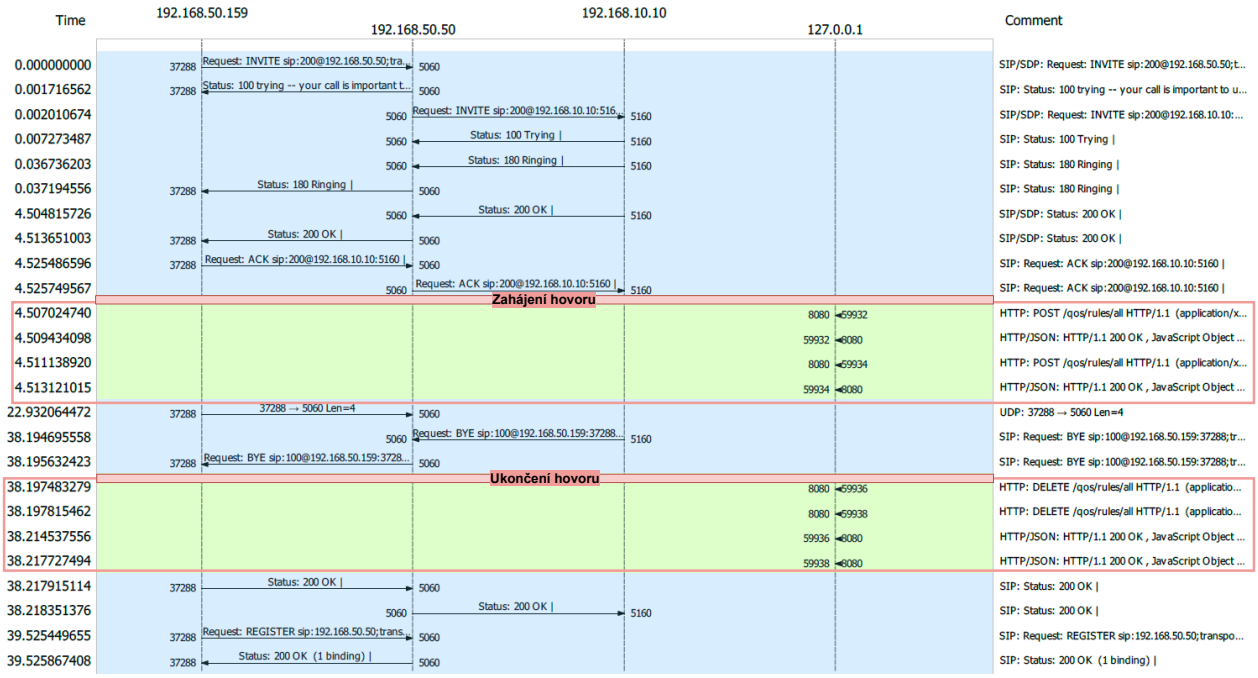
8.1 Dynamická konfigurace

Na obou koncích nasazené SDN sítě byli nakonfigurováni 2 SIP klienti a provedeny zkušební hovory, při jejichž zahájení se dle specifikované konfigurace Kamailia posílaly na SDN kontroler konfigurační HTTP zprávy. Za ukončení hovoru se příslušné konfigurační položky opět smazaly na základě uchovaného identifikátoru. Následuje ukázka zachycených zpráv a flow graf z Wiresharku (8.1), kde lze vidět SIP a HTTP zprávy v časové posloupnosti mezi jednotlivými entitami. V průběhu typického SIP dialogu probíhá dynamická konfigurace prostřednictvím HTTP zpráv. Z architektury sítě plyne, že tyto zprávy jsou vyměňovány lokálně. SIP server Kamailio je dostupný na IP adrese 192.168.50.50, čemuž odpovídá i předcházející konfigurace klientů.

Časová posloupnost u takového zachytávání nemusí být přesná, tedy pořadí zpráv, které jsou časově velmi blízko u sebe nemusí být správné, ovšem pro přibližné znázornění komunikace je tento přístup dostačující.

Následuje zachycená SIP zpráva metody INVITE, ze které si Kamailio navíc z SDP těla uloží port a IP adresu. V tomto případě se jedná o IP adresu 192.168.50.159 a port 48004.

```
INVITE sip:200@192.168.50.50;transport=UDP SIP/2.0
Via: SIP/2.0/UDP 192.168.50.159:37288;branch=z9hG4bK-524287-1---1
   c074514bd03fcd2;rport
Max-Forwards: 70
Contact: <sip:100@192.168.50.159:37288;transport=UDP>
To: <sip:200@192.168.50.50>
```



Obrázek 8.1: Flow graf

```
From: <sip:100@192.168.50.50;transport=UDP>;tag=31cfcb18
Call-ID: cdLKpGzrkczpLgtAvlBnag..
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO,
    SUBSCRIBE
Content-Type: application/sdp
User-Agent: Zoiper rv2.10.12.3-mod
Allow-Events: presence, kpml, talk
Content-Length: 187

v=0
o=Zoiper 1619023613007 1 IN IP4 192.168.50.159
s=Z
c=IN IP4 192.168.50.159
t=0 0
m=audio 48004 RTP/AVP 0 101 8 3
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=sendrecv
```

Za přijetí hovoru, tedy pozitivní odpovědi volajícího 200 OK, se pro oba z účastníků vyšle HTTP zpráva metody POST nesoucí výše uloženou IP adresu a port. Z pozitivní odpovědi 200 OK se dále uloží hodnota parametru `qos_id`.

```
POST /qos/rules/all HTTP/1.1
Host: localhost:8080
User-Agent: kamailio (5.5.0-dev4 (x86_64/linux))
Accept: */*
Content-Length: 103
Content-Type: application/x-www-form-urlencoded
{"match": {"nw_dst": "192.168.50.159", "nw_proto": "UDP", "tp_dst":
    "48004"}, "actions":{"queue": "1"}}

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 228
Date: Wed, 21 Apr 2021 16:46:57 GMT
[{"switch_id": "0000000000000002", "command_result": [{"result": "success
    ", "details": "QoS added. : qos_id=5"}]},
{"switch_id": "0000000000000001", "command_result": [{"result": "success",
     "details": "QoS added. : qos_id=5"}]}]
```

Kamailio za přijetí SIP zprávy metody BYE na SDN kontroler vyšle HTTP zprávu metody DELETE nesoucí odpovídající identifikátor. Stejně jako v předešlé odpovědi 200 OK je to parametr `qos_id` o hodnotě *5*.

```
DELETE /qos/rules/all HTTP/1.1
Host: localhost:8080
Accept: */*
testHeader: header
Content-Length: 15
Content-Type: application/x-www-form-urlencoded
{"qos_id": "5"}

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 226
Date: Wed, 21 Apr 2021 16:47:31 GMT
[{"switch_id": "0000000000000002", "command_result": [{"result": "success
    ", "details": " deleted. : QoS ID=5"}]},
```

```
{"switch_id": "0000000000000001", "command_result": [{"result": "success",
    "details": " deleted. : QoS ID=5"}]}]
```

V průběhu jednoho či více probíhajících hovorů lze vypsat aktuálně nastavená pravidla nástrojem *curl*, například následovně:

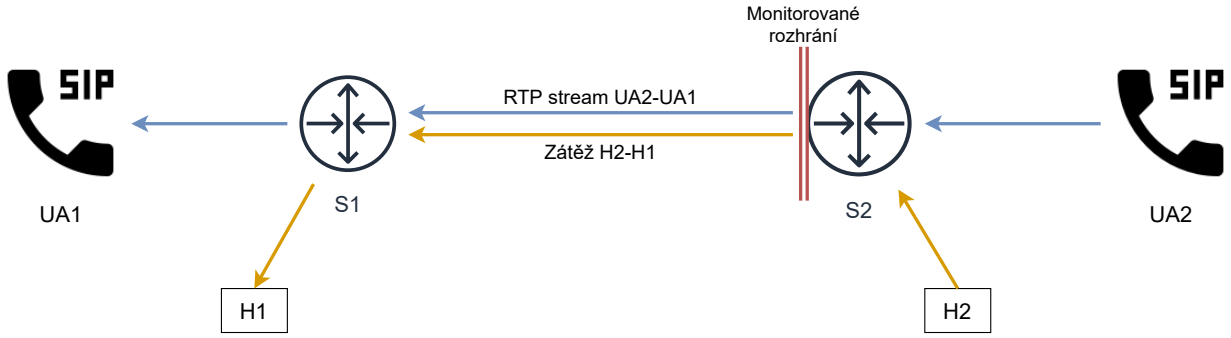
```
# curl -X GET http://localhost:8080/qos/rules/0000000000000001
```

Odpověď obsahuje parametry konfiguračních záznamů přiřazujících dané RTP streamy předkonfigurovaným frontám.

```
[{"switch_id": "0000000000000001", "command_result": [
  {"qos": [
    {"qos_id": 5, "priority": 1, "dl_type": "IPv4", "nw_dst": "
       192.168.50.160", "nw_proto": "UDP", "tp_dst": 55052, "actions":
       [{"queue": "1"}]},
    {"qos_id": 6, "priority": 1, "dl_type": "IPv4", "nw_dst": "
       192.168.10.10", "nw_proto": "UDP", "tp_dst": 5162, "actions": [{"
       queue": "1"}]}
  ]}]}]
```

8.2 Vliv na QoS parametry

Pro ověření vlivu dynamické konfigurace na QoS parametry hovorů se pomocí nástroje *iperf* mezi nasazenými virtuálními hosty vytvoří UDP stream dostatečně zahlcující linku mezi přepínači S1 a S2. UDP stream se tedy bude o nastavenou šířku pásma dělit s odpovídajícím RTP streamem v daném směru. Bez nastavené prioritizace, tedy s výchozí konfigurací SIP serveru, by pro RTP stream na společném výstupním rozhraní mělo dojít k výskytu znatelných hodnot zpoždění a jitteru.



Obrázek 8.2: Zachycení datových toků pro QoS analýzu

Všechna rozhraní, která jsou zde brána v potaz, jsou virtuální. Naskýtá se tedy možnost zachytávat a analyzovat provoz na rozhraní přepínače nástrojem Wireshark. Jeden scénář bude probíhat

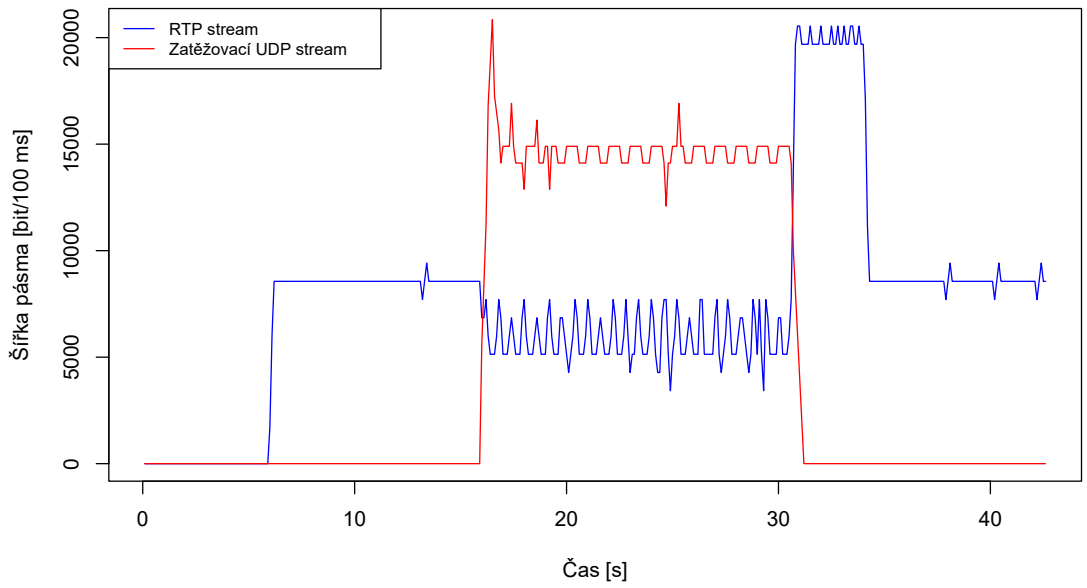
zahájením hovoru mezi UA1 a UA2 a po navázání spojení následovným použitím nástroje *iperf* v terminálu MiniNetu. Zvolil jsem si, že budu monitorovat výstupní rozhraní *s2-eth2* na přepínači S2, což znamená, že zatěžující provoz bude nutné generovat z H2 na H1. Maximální kadence generování dat se bude rovnat nastavené maximální šířce pásma na rozhraní, které chceme pozorovat.

Druhý scénář bude probíhat identicky, až na výměnu konfiguračního souboru Kamailia, které bude provádět pouze jeho základní funkce, tj. nebude probíhat dynamická konfigurace QoS politik.

```
h2 iperf -c 172.16.20.10 -p 5001 -u -b 200k
```

Grafy znázorňující rozdělení přenosového pásma byly pro lepší rozlišení zpracovány v 100 ms intervalech (v jednotkách bit/s je to tedy 10-krát více). V programu RStudio byla na jednotlivé křivky aplikována funkce pohyblivého průměru SMA.

Bez nastavení prioritizace se šířka pásma rozdělí v poměru k jednotlivým datovým tokům. Jelikož byla překročena maximální šířka pásma na rozhraní, přenos obou datových toků je omezen a dochází k zaplňování fronty (bufferu). Po úplném přenesení zatěžovacího UDP streamu následuje plné využití šířky pásma samotným RTP streamem. To značí vyprázdňování zahlcené fronty, což v daném časovém úseku ovlivní i další QoS parametry, jak lze vidět v následujících grafech.



Obrázek 8.3: Rozdělení šířky pásma ve výchozí konfiguraci

V dalším grafu (8.4) je znázorněna hodnota zpoždění RTP paketů vypočtená na základě očekávaného času přijetí. Očekávaný čas přijetí je odvozen ze sekvenčního čísla RTP hlavičky, frekvence vysílání paketů (packet rate), což je pro využitý kodek G.711 50 paketů za sekundu (pps), a sumy

rozdílů mezi časy zachycení po sobě jdoucích paketů Δ. Tato metoda by v některých případech nemusela být přesná, ale za předpokladu, že je počáteční hodnota zpoždění zanedbatelná, měla by být zanedbatelná i ona nepřesnost.

$$t_k = \sum_{n=k_0}^{k} \Delta_n - \frac{1000}{r} \times (k - k_0)[ms]$$

Kde:

t_k = zpoždění daného paketu

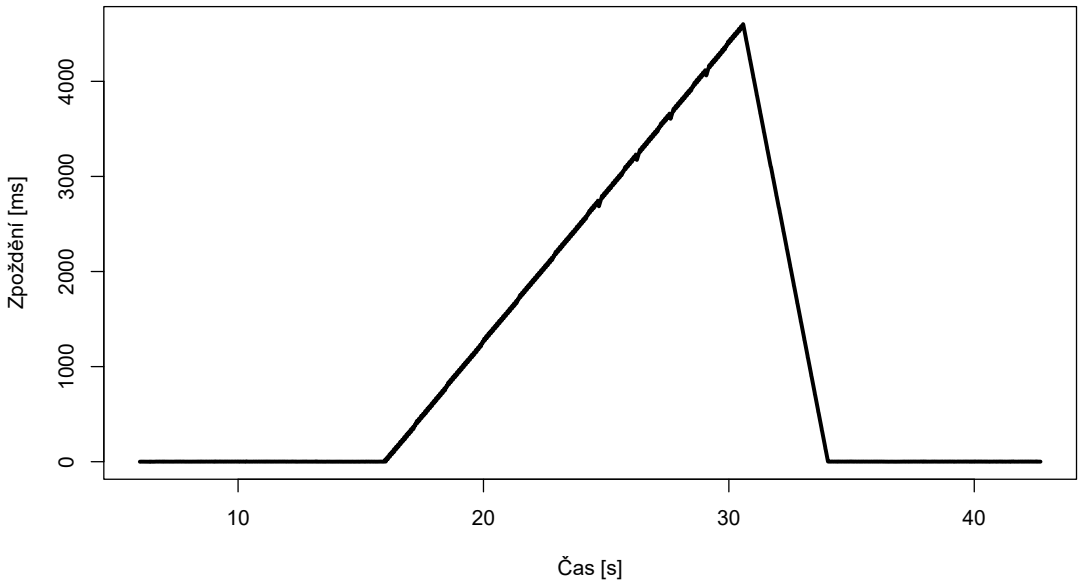
k = sekvenční číslo daného paketu

k_0 = sekvenční číslo prvního paketu RTP streamu

Δ = rozdíl času přijetí oproti předchozímu paketu daného RTP streamu

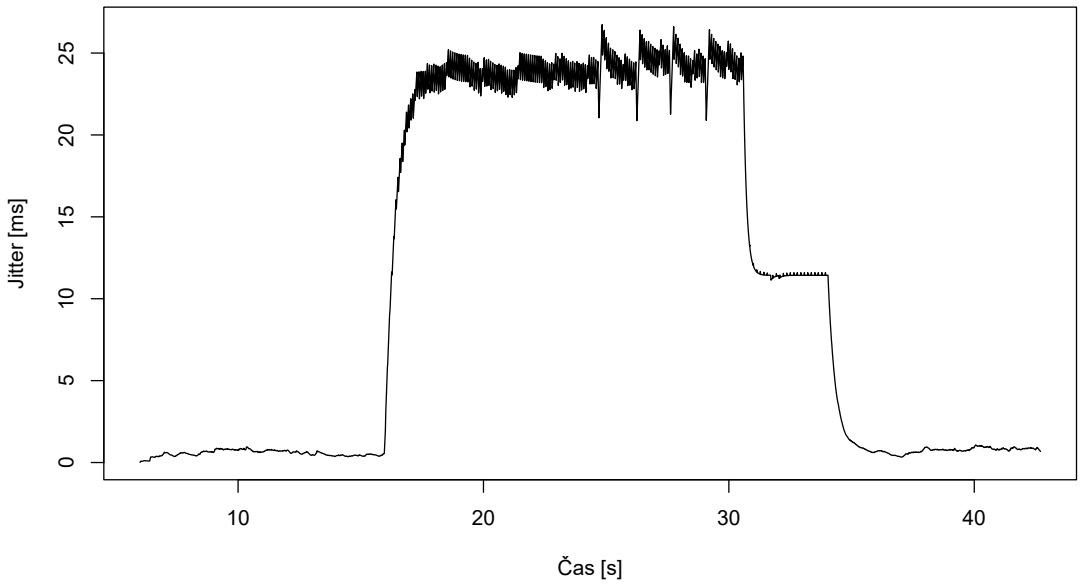
r = frekvence paketů RTP streamu (packet rate [pps])

Zpoždění je zanedbatelné až do bodu zahlcení, při kterém lze v grafu vidět lineární nárůst, a následně lineární pokles, což zrcadlí zmíněný interval, ve kterém RTP stream využívá celou šířku pásma rozhraní.



Obrázek 8.4: Zpoždění RTP streamu na měřeném rozhraní ve výchozí konfiguraci

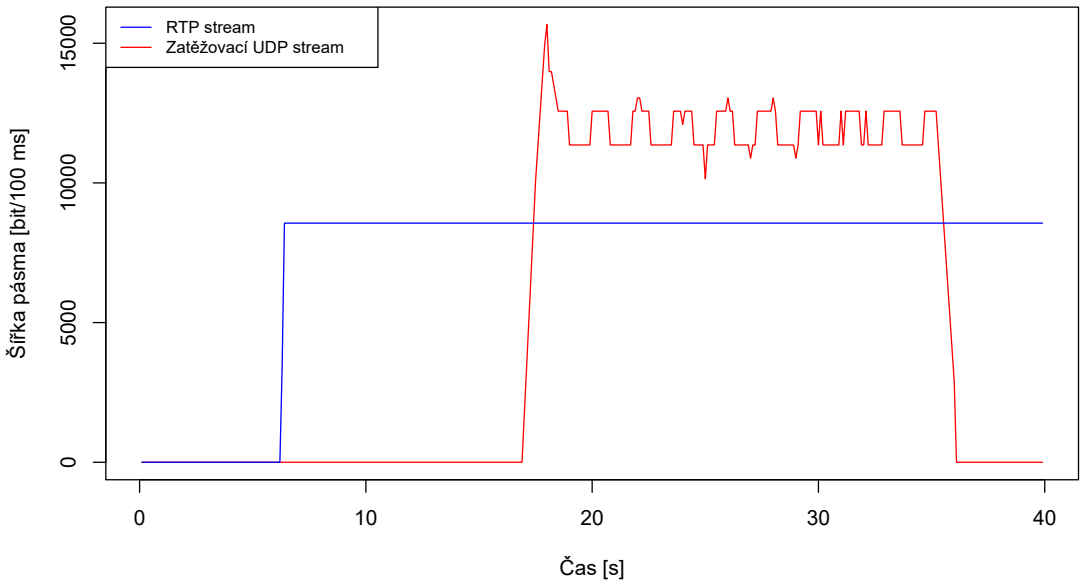
Hodnota jitteru v grafu 8.5 je podobně jako u zpoždění zanedbatelná do výskytu zátěže a po uvolnění fronty.



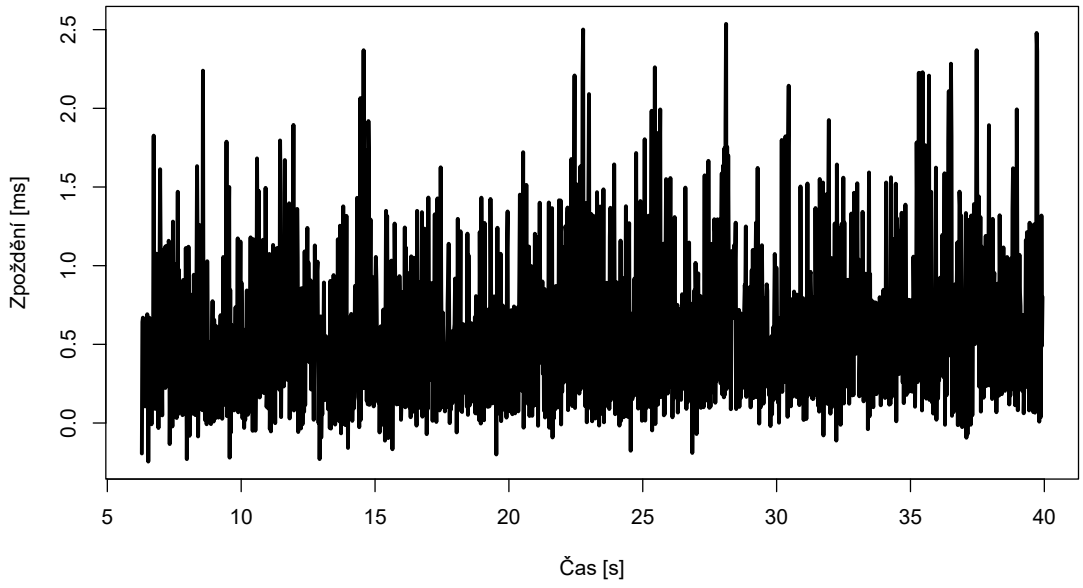
Obrázek 8.5: Jitter RTP streamu na měřeném rozhraní ve výchozí konfiguraci

S nasazeným systémem dynamické konfigurace nastává výrazná změna. Pakety prioritizovaného RTP streamu jsou posílány přednostně a využívaná šířka pásma v tomto případě není omezena, ovšem byla by při překročení nastavené minimální dostupné šířky pásma nastavené fronty. Přenos zatěžovacího streamu má i delšího trvání než v předchozím případě, přestože jeho parametry jsou totožné. To plyne z faktu, že v tomto případě je do bufferů ukládána jeho větší poměrná část, kterou je potřeba přenést později po ukončení vysílání dat na zdroji.

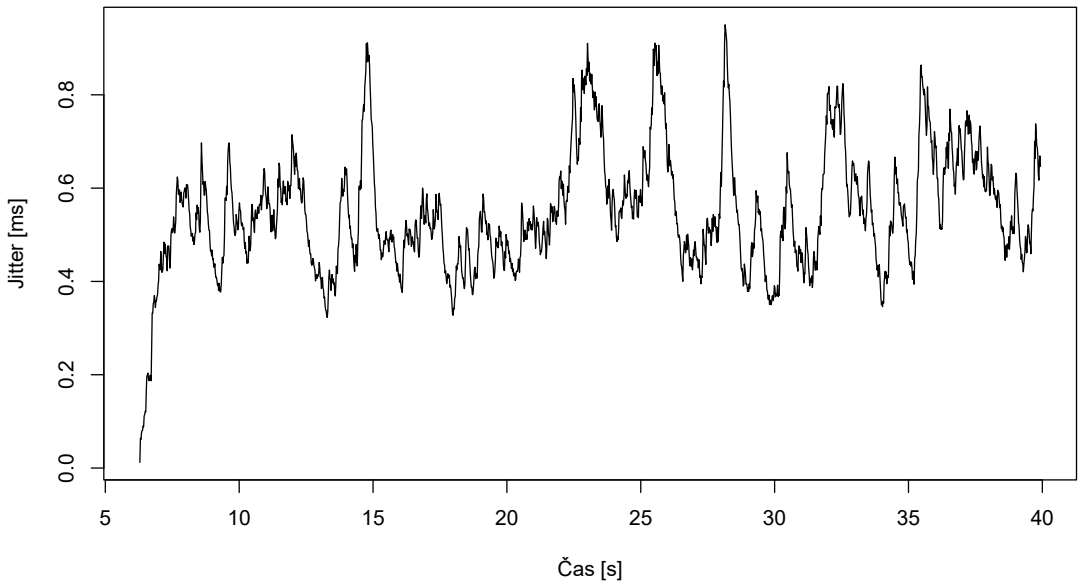
Zpoždění i jitter se v tomto případě pohybují v řádech desetin až jednotek milisekund, což se dá považovat za zanedbatelné hodnoty.



Obrázek 8.6: Rozdělení šířky pásma s dynamickou konfigurací QoS politik



Obrázek 8.7: Zpoždění RTP streamu na měřeném rozhraní s dynamickou konfigurací QoS politik



Obrázek 8.8: Jitter RTP streamu na měřeném rozhraní s dynamickou konfigurací QoS politik

Kapitola 9

Závěr

V teoretické části této práce byly popsány koncepty a protokoly, které figurují v následné praktické části. Pro návrh výsledného systému se jako stěžejní prokázaly znalosti z oblasti SIP architektury, která zde slouží jako iniciátor dynamické konfigurace SDN prvků.

Pracovní topologie byla navržena zejména s ohledem na ovlivnitelnost QoS parametrů přenášených hovorů. V sestaveném prostředí, skládajícího se ze SIP serveru Kamailio a SDN sítě s kontrolerem Ryu propojující telefonní klienty, lze tedy pozorovat vliv dodané konfigurace na kvalitativní parametry probíhajících hovorů.

Grafy zobrazující hodnoty zpoždění, jitteru a šířky pásma ukazují, že při nastavení dynamické prioritizace hovorů je omezen pouze zatěžovací datový tok. Naproti tomu v případě bez dynamické prioritizace dochází za stejných podmínek k výraznému zhoršení kvalitativních parametrů hovoru. Toto srovnání dokazuje funkčnost navrženého systému a potvrzuje jeho pozitivní vliv na kvalitativní parametry přenášených hovorů.

Jako hlavní výhoda SDN sítí se v této práci ukázala zejména centralizovaná konfigurace dílčích síťových prvků, jež umožňuje jednotnou konfiguraci bez hlubší znalosti topologie sítě z pohledu konfigurující entity.

Demonstrované postupy by bylo možné využít pro jinou práci podobného charakteru a obohatit je například o dynamické nastavení front nebo o mechanismus konfigurace na straně kontroleru. V určitých implementacích by bylo vhodné zde uvedený systém aplikovat tak, aby zaručoval přednost přenosu konkrétních hovorů jako jsou např. tísňová volání.

Literatura

[1] BARREIROS Miguel, LUNDQVIST Peter. *QOS-ENABLED NETWORKS*. UK: John Wiley & Sons Ltd, 2016, 2. edice. ISBN: 9781119109105

[2] VOZNAK Miroslav. *Technologie a protokoly multimediálních komunikací pro integrovanou výuku VUT a VŠB-TUO*[CD-ROM]. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2014. ISBN 978-80-248-3326-2.

[3] GORANSSON Paul, BLACK Chuck, CULVER Timothy. *Software Defined Networks: A Comprehensive Approach*. UK: Morgan Kaufmann, 2016, 2. edice. ISBN: 978-0-12-804555-8

[4] RYU project team. *Ryubook* [online]. Ryu, 2014. Dostupné na: https://osrg.github.io/ryu-book/en/html/

[5] Open Networking Foundation. *OpenFlow Switch Specification* [online]. Open Networking, 2012, verze 1.3. Dostupné na: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[6] ROSENBERG J., SCHULZRINNE H., CAMARILLO G., et al. *SIP: Session initiation protocol* [online]. Request for Comments 3261, Červen 2002. Dostupné z IETF: https://tools.ietf.org/html/rfc3261

[7] JANAK Jan. *SIP Proxy Server Effectiveness*. Praha: Česká Technická Univerzita - Fakulta elektrotechniky - Oddělení počítačových věd, 2003.

[8] MIERLA Daniel-Constantin, MODROIU Elena-Ramona. *Kamailio SIP Server v3.2.0 Development Guide* [online]. ASIPTO, 2011. Dostupné z Asipto: http://www.asipto.com/pub/kamailio-devel-guide/

[9] Kamailio. *Kamailio SIP Server v5.3.x (devel): Core Cookbook* [online]. Kamailio, 2019. Dostupné z Kamailio SIP Server Wiki: https://www.kamailio.org/wiki/cookbooks/devel/core

[10] Kamailio. *Kamailio - Getting Started Guide* [online]. Kamailio, 2015. Dostupné z Kamailio SIP Server Wiki: https://www.kamailio.org/wiki/tutorials/getting-started/main

[11] ZEMAN Dalibor. *Nasazení technologie WebRTC s využitím SIP Proxy Kamailio* [online]. Ostrava, 2019. Dostupné z: http://hdl.handle.net/10084/136345. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava.

[12] MIERLA Daniel-Constantin. *SDPOPS Module* [online]. Kamailio. Dostupné z Kamailio SIP Server Dokumentace: https://kamailio.org/docs/modules/4.1.x/modules/sdpops.html

[13] CABIDDU Federico, VACCA Giacomo, OUDOT Camille. *HTTP_ASYNC_CLIENT Module* [online]. Kamailio. Dostupné z Kamailio SIP Server Dokumentace: https://www.kamailio.org/docs/modules/devel/modules/http_async_client.html

[14] JOHANSSON E. Olle, HEINANEN Juha, BOCK Carsten. *http_client* [online]. Kamailio. Dostupné z Kamailio SIP Server Dokumentace: https://www.kamailio.org/docs/modules/devel/modules/http_client.html

[15] IANCU Bogdan-Andrei, BOCK Carsten. *dialog Module* [online]. Kamailio. Dostupné z Kamailio SIP Server Dokumentace: https://www.kamailio.org/docs/modules/devel/modules/dialog.html

Příloha A

Přílożené soubory

Součástí této práce jsou soubory rozdělené následujícím způsobem:

- **Kamailio** - konfigurační soubory Kamailia
- **Konfigurační zprávy** - zachycená komunikace ze sekce 8.1
- **Testovací hovory** - zachycená komunikace ze sekce 8.2